

Xpirit Magazine

Sam Guckenheimer
DevOps for Mobile Apps

René van Osnabrugge
How Docker will change
Microsoft development

Alex Thissen
.NET Core: a familiar
and different .NET Platform

Other articles are: ■ Scaling Scrum Professionally using Nexus and Visual Studio Team Services - Jesse Houwing ■ Using the Actor Model to create distributed applications with Akka.NET - Pascal Naber ■ Enhancing your insights with Power BI - Jasper Gilhuis ■ Integrating Protractor UI testing in Visual Studio, TFS and VSTS - Marcel de Vries ■ High availability and disaster recovery in Azure - Loek Duys ■ Installing Cloudera on Azure - Alexander Bij / Tünde Alkemade ■ Building a Robot Kit with a Raspberry PI 2 and Windows 10 IoT Core - Marco Mansi



Get to market faster with DevOps

Find your opportunities for growth and see how your progress aligns with Microsoft's seven key habits of DevOps success with a free, 10-minute self-assessment.

Take the free assessment now
<http://aka.ms/devops>

Colofon

Edition:

Xpirit Netherlands BV

No. 2 • February 2016

Editorial Office:

Xpirit Netherlands BV

This magazine was made with the help of Pascal Greuter, Sam Guckenheimer, Alex Thissen, Rene van Osnabrugge, Jesse Houwing, Pascal Naber, Jasper Gilhuis, Marcel de Vries, Alexander Bij, Tünde Alkemade, Marco Mansi

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

©2016 All rights reserved.

No part of the contents of this magazine may be reproduced or transmitted in any form or by any means without the written permission of the Xpirit Netherlands BV.

All trademarks are property of their respective owners.

Advertisement

Microsoft	2
Xamarin	17
Xebia University	24
Go Data Driven	55

Welcome to Xpirit Magazine #2!



We love to celebrate our first year of operations with you in this second edition of Xpirit magazine with great articles on insights that we have gained last year, as well as the emerging trends that we are seeing. Since our startup in 2014, we've worked with over 50 organizations, ranging from independent software vendors to international utilities, finance institutions, manufacturers and insurance companies. We helped them deliver real productivity gains and take their IT systems into the next era through our full cloud transformation services and bringing their application life cycle management under control. Along the way, our team of leading IT specialist have been travelling the world to share and extend our expertise in these latest technologies. In this edition we composed a great line-up of articles on topics we encountered and that show trends in the changing eco-system for developers on the Microsoft platform.

The success of our business and the compliments we've received from the industry have stemmed from our core principles: People First, Quality Without Compromise, Customer Intimacy, and Sharing Knowledge. By working closely with our customers and really understanding their business and technology challenges, we've been able to turn our customers' needs into concrete solutions and clear operational roadmaps using the latest Microsoft technologies. Almost all Xpirit's customers who have been working with us since our startup are still with us today.

At Xpirit, we work with the best people in the world to achieve outstanding results. We are extremely proud of what our close-knit team of IT specialist has achieved. Xpirit, as proudly part of Xebia Group, is able to roll out innovative solutions for a diverse clients in a much broader range than just Microsoft technologies. In our partnership with Xebia, we have been able to apply our expertise in Microsoft and embrace open source to develop and deliver a wider range of business services at our customers as well.

This magazine will give you a fresh set of ideas that you can use daily and that will have an impact on your organization. The adoption of Docker is really important for .NET developers as is the new .NET Core and ASP.NET implementation. There is also hands-on information on Cloudera on Azure, how to implement disaster recovery for cloud-hosted web applications, and a tutorial that shows how to get started with actor model development using Akka.NET. PowerBI is bringing big data within reach for those who do not have a background in data science. In addition, we show you how to create an extension to test adapters by incorporating new types of tests into Visual Studio. In the downloadable version of the magazine you will also find an additional article on running Windows IoT Core on Raspberry Pi, which we encourage you to try out for fun coding.

It has been an exciting startup year, and we are looking forward to even bigger things next year. We wish you an informative read, and look forward to working with you during the coming year.

Pascal Greuter
Managing Director Xpirit





Table of contents

■ Welcome to Xpirit Magazine	3
■ Table of contents	4
■ DevOps for Mobile Apps	5
■ .NET Core: a familiar and different .NET Platform	7
■ How Docker will change Microsoft development	13
■ Scaling Scrum Professionally using Nexus and Visual Studio Team Services	18
■ Using the Actor Model to create distributed applications with Akka.NET	25
■ Enhancing your insights with Power BI	34
■ Integrating Protractor UI testing in Visual Studio, TFS and VSTS	40
■ High availability and disaster recovery in Azure	47
■ Installing Cloudera on Azure	52
■ Building a Robot Kit with a Raspberry PI 2 and Windows 10 IoT Core ...	57

DevOps for Mobile Apps

Three technology trends have dominated this decade. First, applications have become connected across mobile, cloud services, and sensors, to the point where each part depends on connectivity to deliver a complete experience. Second, DevOps practices and tools have revolutionized the pace and quality of service delivery. Third, mobile devices have become the primary means of consumer and employee access to these connected systems, but their practices and tools have lagged behind impediments imposed by distribution and diversity.

According to ComScore, mobile became to majority source of web traffic in mid 2014.¹ And Gartner believes that 60% of all business processes will be optimized for mobile by 2020². This indicates a permanent shift in technology usage and a permanent change in the way software applications are being built.

DevOps evolved for web applications and defined a New Normal for software teams. Lean management and continuous delivery practices created the conditions for delivering value faster, sustainably. For example, high-performing IT organizations report 30x more frequent deployments with 200x shorter lead times, 60x fewer failures and 168x faster recoveries³.

Unfortunately, these deployment pipelines are tuned for centrally administered servers. When servers directly push data directly to the web, it is relatively easy to monitor both the quality of service and customer behavior. Errors can be corrected, performance can be improved, and experiences can be enhanced with new server deployments alone.

Unfortunately, deployment to mobile devices is typically gated behind app stores, managed by the device vendors. This creates the need for a dual release pipeline: one path for the service side and one for the mobile devices.

What is needed to make DevOps for mobile applications effective? The same kind of practices that apply for services apply for devices, tuned to the differences in technology:

1. A release pipeline. Continuous integration should produce mobile packages for the appropriate operating systems (e.g. iOS and Android) and versions and launch automated testing of standard configurations and scenarios as part of a continuous deployment pipeline. Integration with device clouds can allow

automation of testing across multiple form factors as part of the standard release process.

2. Distribution of "beta" releases. Especially for consumer apps, where star ratings are so important, beta releases need to get thorough exploratory testing by cohorts of real users. This requires the ability to have users directly load the beta versions or to target cohorts with packages that have not had to go through the official stores. This needs to be an iterative process that allows as many redistributions as needed.

3. Test coverage should be tracked through all these phases, not just against code, but also against all the device, OS, carrier, network and language configurations. For Android, with thousands of devices, this matrix expands to millions of configurations.

4. Remediation. When a user experiences a crash (or a performance bottleneck), you need to learn of this instantly. Your application's crashes need to come back to you as stack traces, matched to the right version of source code, with symbols and environment details, so that you can fix them unambiguously.

5. Whole lifecycle. Any live error that the application generates should be tracked as a bug so that you can tell whether you have remediated it and create a regression test with the code to prevent its recurrence.

6. Feedback. Experimentation is the essential juice of a modern lifecycle, as you always make your apps better for users. You want to be sure that your cohorts of testers can give you rich feedback on their experiences so that you can make the apps better.

¹ <http://www.comscore.com/Insights/Blog/Major-Mobile-Milestones-in-May-Apps-Now-Drive-Half-of-All-Time-Spent-on-Digital>

² Gartner AADI Keynote, December 2015

³ 2015 State of DevOps Report

In a DevOps world, these practices are taken for granted, but they have been very hard to implement for mobile. This has motivated Microsoft to introduce HockeyApp to extend DevOps to mobile apps too. Together with Visual Studio Team Services and the developer's IDE, it builds the foundation for the Mobile DevOps cycle.

Gene Kim has suggested that DevOps goes through a progression of the "Three Ways".⁴ The first is the automation of the release pipeline to allow the continuous flow from Development to

Operations, or in this case, Mobile Deployment, as that is the initial bottleneck. The second is the feedback from Deployment to Development, to allow Development to become increasingly aware and responsive of operational issues. The third is the continual amplification of the feedback loops, so that improvements can flow faster and faster. HockeyApp is one tool that seeks to close these gaps in practice from the server-side to mobile development, so that your team can apply the best learnings of DevOps to both halves of the application and both parts of the release pipeline.

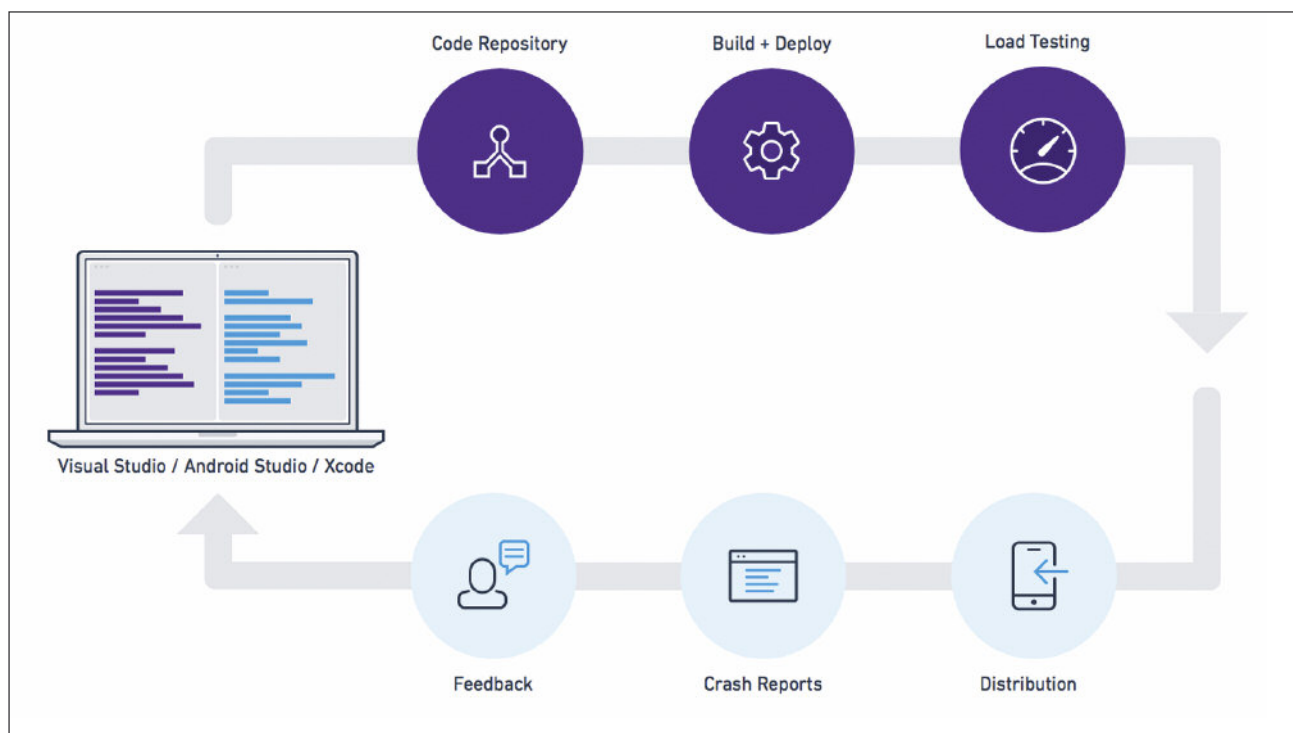


Figure 1



SAM GUCKENHEIMER

**GROUP PRODUCT PLANNER
MICROSOFT**

Sam has 25 years of experience as an architect, developer, tester, product manager, project manager and general manager in the software industry in the US and Europe. He holds five patents on software lifecycle tools. A frequent speaker at industry conferences, Sam is a Phi Beta Kappa graduate of Harvard University.

⁴ Gene Kim, Kevin Behr and George Spafford, *The Phoenix Project* (Portland: IT Revolution Press, 2013)

.NET Core: a familiar and different .NET Platform

The ASP.NET team has taken .NET in a new direction in order to achieve the vision for a new cloud-optimized web framework. The team envisioned a redesigned framework to overcome the limitations of the original 15-year-old .NET Framework 4.5 and a modern cross-platform story to be able to quickly adapt to the changing Web. The Web stack was redesigned and rebuilt nearly from scratch and along with it, the .NET Framework. Now, more than a year down the road, the .NET rework has been adopted by Microsoft to serve as the future of a brand new .NET Platform, starring .NET Core.

More than meets the eye

ASP.NET introduces a new implementation of .NET Framework libraries and new runtimes plus additional tooling. Together these are named .NET Core. A casual glance at .NET Core might give the impression that this is merely a trimmed-down, lightweight version of the original .NET Framework. To some degree that is true. A closer look at .NET Core reveals a complete overhaul of the way .NET is engineered, implemented, and offered as a modern platform to build a variety of application types. The new version of .NET is called .NET Core 1.0. This new version number of 1.0 is chosen to indicate the revolution of .NET, as opposed to a natural evolution of the previous iterations of 1.0 through 4.6, that 5.0 would suggest. The framework libraries largely remain unaffected from a functional perspective and your current knowledge of programming for .NET is still valid. Underneath the familiar surface of .NET is a new runtime, NuGet as a packaging format for the distribution of .NET libraries, and brand new tooling to manage the globally installed runtimes for .NET. All in all, a lot of significant changes that deserve attention that goes beyond .NET Core.

A shift in strategy

The most obvious changes are easy to see and have been advocated by the ASP.NET and .NET team: the new .NET Core is now a much smaller and self-contained framework and runtime for applications. This means that .NET Core does not have to be installed globally on machines that use a particular version. Moreover, multiple versions can exist side-by-side and be deployed as part of the application. Also, .NET Core can be used on multiple operating systems and allows Windows, Linux, FreeBSD and MacOS to host applications built on top of .NET Core.

The small footprint and isolated deployment combined with the cross-platform story opens up new scenarios for .NET applications. One important scenario is the ability to create and run Docker containers for Linux and Windows with the appropriate version of .NET Core. This makes .NET a valid choice when adopting a container approach for building and hosting your applications.

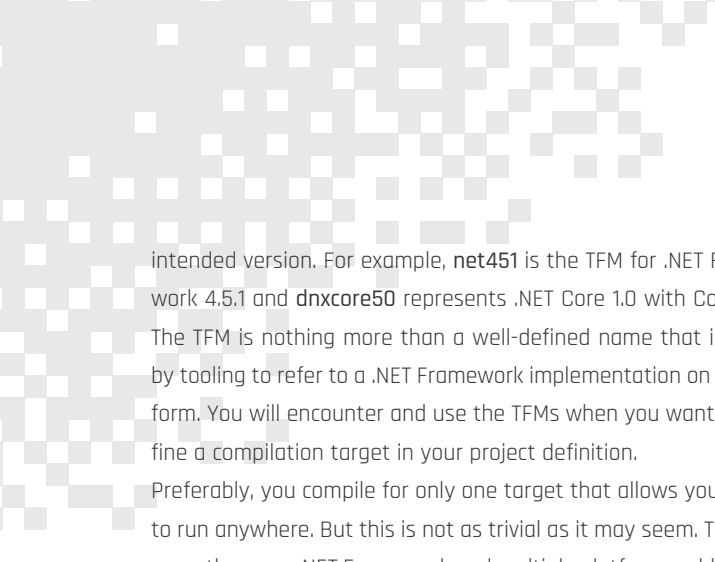
Another important change is the decoupling of versions of the .NET Runtime from the framework. This means that multiple versions of runtimes for different operating systems and processor architectures can run your executables and libraries. Your binaries might be based on .NET Framework libraries that are not necessarily of the same version as the runtime, nor compiled specifically for a platform on which the runtime exists. This does require a new approach to the way libraries are compiled and made compatible across runtimes and platforms.

Multi-targeting

Traditionally the .NET Framework Software Development Kit (SDK) had a specific version targeting a single version of the .NET Runtime and Framework. Later editions of tooling such as Visual Studio introduced multi-targeting. This allowed a code project for an application or library to choose the version of the framework and runtime they depend and build upon. Still, the compiler was able to build binary images solely for a single framework version, one at a time.

With .NET Core come new compilers such as Roslyn, the Compiler-as-a-Service facilities of .NET that allows actual compiling your source code for multiple framework targets.

Each of these targeted frameworks is denoted by a Target Framework Moniker (TFM) that is a short symbol representing the



intended version. For example, **net451** is the TFM for .NET Framework 4.5.1 and **dnxcCore50** represents .NET Core 1.0 with CoreCLR. The TFM is nothing more than a well-defined name that is used by tooling to refer to a .NET Framework implementation on a platform. You will encounter and use the TFMs when you want to define a compilation target in your project definition.

Preferably, you compile for only one target that allows your code to run anywhere. But this is not as trivial as it may seem. There is more than one .NET Framework and multiple platforms add to the complexity.

Platforms and frameworks

The .NET Platform has long been a broad and general term to capture several elements that together form a platform for building and running applications. The .NET Framework, the Software Development Kit, the Windows desktop and the server operating system were the most important elements. More operating systems were supported at a later stage, not only Windows, but also Windows Phone, Linux and OS X through Mono (a partial cross-platform port of part of the full .NET framework). Most recently the Windows 10 wave with Universal Windows Applications spanning across desktop, server, mobile and Internet of Things (IoT) devices were added. Each of these operating systems are separate .NET Platforms with their own version (or multiple versions) of a .NET Framework. This year .NET Core will be added to this list of platforms.

.NET Platform Standard

Transitioning your code to .NET Core requires your code and the libraries it depends upon to be available and compatible with the environment you run in. If you use an open-source library, it might use certain APIs that are not. For example, a library could have references to assemblies from the full .NET Framework 4.0 that have not been ported to run cross-platform on Linux or OS X. When the code tries to call such assemblies they are not available, essentially causing your code to not work. Luckily Microsoft has come up with a way to indicate how much of .NET is supported where. Staying within these boundaries will guarantee your code to run wherever you intend it to run.

Originally, Portable Class Libraries (PCL) was the first solution to these challenges of a variety of different platforms, with their different capabilities and specific .NET Framework implementations. PCL introduced a reduced subset of the .NET Framework that works on each of the compatibility-targeted .NET platforms.

This subset is based on the lowest common denominator of support per .NET platform, so there is a guarantee that it will stay within the boundaries of the supported functionality in the .NET API. The .NET API per platform is defined by a set of assemblies that define the shape of .NET types without implementation. These so-called 'reference assemblies' form the build-time contracts of .NET Framework per platform.

The approach of PCL to provide cross-framework compatibility has evolved. The **.NET Platform Standard** provides an open-ended way to represent binary portability across platforms. It is a standard that uses versioned sets of reference assemblies to define the surface API of the .NET Framework that is available to library developers on each of the platforms. The targeted platforms themselves have an actual implementation per reference assembly that you as a developer can use in your code. Reference assemblies are explicitly used to define a contract for the API surface and are now referred to as 'contract assemblies'. Contrary to PCL, which defines the included platforms at build time, a .NET Platform Standard version defines the surface API that must be supported. This allows newer platforms not available at the time of build to also use the libraries compiled against a specific version of a .NET Platform Standard reference set, without having to recompile the app to support the new platform.

The .NET Platform Standard defines multiple versions of such reference sets of contract assemblies, named .NET Standard 1.0, 1.1 up to 1.4. More will be added in the future when new APIs are created and the reference contracts in the surface API change in an incompatible manner. Each of the reference sets promise and require compatibility with the specific .NET Platforms and frameworks. This has the important implication that platforms might be dropped from support for newer version of the .NET Platform Standard. Fewer platforms are supported, but a bigger .NET surface API will be available to those that remain.

The following matrix shows the supported .NET platforms for each of the current .NET Platform Standard versions. Start from a .NET Platform Standard version (e.g. version 1.3) and read that column downwards. Each colored horizontal block that crossed the column indicates compatibility. A library targeted at .NET Platform Standard 1.3 can run on .NET 4.6 and 4.6.x, UWP 10.0 and .NET Core 5.0.

		.NET Platform Standard				
Target Platform Name	Alias /TFM	1.0	1.1	1.2	1.3	1.4
.NET Framework	net			4.6.x		
		4.6				
		4.5.2				
		4.5.1				
		4.5				
Universal Windows Platform	uap	10.0				
Windows	win	8.1				
		8.0				
Windows Phone	wpa	8.1				
		8.0				
Windows Phone Silverlight	wp	8.1				
		8.0				
DNX Core	dnxc	5.0				
Mono/Xamarin Platforms		*				
Mono		*				

Figure 1

* indicates that the platform version that is supported for Mono and Mono/Xamarin is yet to be determined.

The surface API defined in the contract assemblies of the .NET Standard 1.0 is small enough to be available on the full .NET Frameworks 4.5, and later on Windows, the Universal Windows Platform, Windows Phone 8.0 and 8.1, .NET Core and Mono. Higher versions of the .NET Standard have a bigger surface API but are only available on the newer platforms (since the older ones will not be updated anymore). .NET Core is the new kid on the block, and offers an implementation for the highest .NET Standard with the broadest surface API.

Once you create a library and target a Platform Standard of a certain version, you know it is compatible and able to run on the supported .NET Platforms. Like all platforms, the standard version also has a TFM: `netstandard1.0`, `netstandard1.1` to `netstandard1.4`. You need these monikers when you create a library assembly yourself and want to have a certain level of compatibility.

The .NET Standard Library

Circling back to .NET Core: the .NET Framework implementation targeted at the .NET Core Platform is called CoreFX. It has an implementation according to .NET Standard 1.4. The list of contract assemblies in 1.4 is captured in the .NET Standard Library: a combination of assemblies specific to the Platform Standard for multiple platforms and platform-agnostic assemblies that are the same across platform versions. The following figure shows the platform structure.

The center of the figure contains the Platform Standard libraries, which comprises the implementation according to a .NET Platform

Standard as outlined before. However, some assemblies are implemented as part of the Platform specific runtimes, such as the full .NET Framework set or .NET Core. Usually these assemblies contain operating system calls that are abstracted away in .NET libraries. An example would be System.IO that has a different implementation for Windows, Linux and OS X. These 'anchored' assemblies can only be updated by updating the particular framework. Some assemblies are agnostic of the platform, such as System.Linq. Such assemblies build upon the Platform Standard and framework assemblies, but

are not different for different versions, platforms or operating systems.

The .NET Standard Library is the combination of the .NET Platform Standard designated contract APIs and the platform agnostic ones. By referencing that library, you know that your code uses a .NET API that allows you to target, compile and run your code on Full .NET, .NET Core or Mono. You will start seeing the related NuGet package appear as a convenient way to reference .NET Platform Standard implementations.

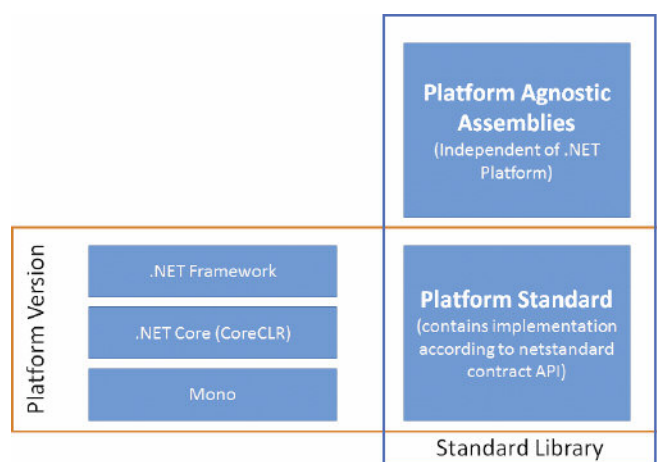


Figure 2

Inside the core

.NET Core is a cross-platform and open-source implementation of .NET. It consists of three components:

- CoreFX: .NET Core base class libraries including implementation of .NET Standard Library
- Multiple compilers: RyuJIT, .NET Native, Roslyn and LLILC
- Two runtimes: CoreCLR and CoreRT
- Command-line interface tooling

These components show the clear separation of the .NET Core Framework and the runtimes. It is also visible in the ownership and location of the implementations. Each of the components are separate entities with their own Git repository, which allows each individual component to evolve separately while providing a .NET Platform together.

We will not drill into the different compilers that exist for .NET Core, but suffice it to say that depending on your scenario, you can choose for compilation to Intermediate Language (IL), native code through C++ code generation. Ahead-of-Time (AOT) compilation for creating single binaries that contain both your compiled code together with the required runtime and libraries is also available.

Of the two runtimes CoreCLR is the most important and complete one. It is a Common Language Runtime much like that of the full .NET Framework. The main difference is the cross-platform (Windows, Linux, OS X and FreeBSD) implementation that was created for CoreCLR. CoreRT is a runtime for code compiled with .NET Native as it requires a different way of executing.

.NET Core tooling

Since .NET Core is cross-platform over Linux distributions, OS X, FreeBSD and Windows, the development experience cannot rely on the general availability of a rich Integrated Development Environment (IDE) such as Visual Studio 2015. Microsoft has created Visual Studio Code as a lightweight cross-platform IDE for Linux and Mac. Command-line tooling is the common ground for development on the various operating systems. This strategy allows you to pick your favorite environment (VI, Emacs, Notepad, Sublime Text or Visual Studio Code on your OS, or Visual Studio 2015 on Windows) for working with code and perform operations from the command-line.

.NET Core Command-Line Interface

.NET Core includes a set of stand-alone command-line interface (CLI) tools that allow you to complete various tasks related to .NET Core components. Currently, the CLI tooling is a separate download and installation.

The previous incarnation of this tooling used **dnu.cmd** and **dnx.exe** as two separate pieces of command-line tooling. The .NET Development Utility (dnu) was for compilation and packaging, while the .NET Execution Environment (dnx) was the bootstrapper for running .NET Core applications. Each of these tools is now bundled in a new CLI tool **dotnet.exe**.

dotnet is the CLI for performing .NET Core related operations. It is an extensible entry point to various operations related to .NET Core projects and source code. **dotnet.exe** is a driver that takes a command to indicate the actual operation. A couple of examples will show what it can do:

Command	Operation
dotnet new	Scaffold a new empty .NET project
dotnet restore	Restores dependencies of projects
dotnet build	Performs a build of a project's source code
dotnet pack	Creates a NuGet package for the compiled binaries
dotnet repl	Interactive REPL session
	(Read, Eval, Print, Loop)

Figure 3

Each of these commands can accept a number of arguments if different options or behavior are needed. The **dotnet** tool has built-in documentation for each of the commands, which can be viewed by passing the **--help** switch for the particular command.

Each of the commands is actually a separate executable that is spawned from the **dotnet.exe** tool. The convention is that the executable for a command is named **dotnet-command.exe**, such as **dotnet-build.exe**. The commands can be executed directly, but the preferred way is to use the top level driver **dotnet**.

Initially you typically run **dotnet** from an empty directory with the name of your .NET project. The folder name will be the default name of your assembly. After that you can use Visual Studio 2015, Visual Studio Code or your preferred code editor to program your application or library. The two Visual Studio editions have built-in support for the command-line tooling as part of their build system. At the time of writing this article, this is still the precursor tooling of **dnx.exe** and **dnu.cmd**, but this will probably be updated to support the **dotnet.exe** tooling.

A short walk-through of Hello World in .NET Core

Let's have a look at .NET Core by creating and inspecting a simple Hello World application. Make sure you have .NET Core installed on your operating system. Instructions for doing this can be found on the GitHub repository for .NET Core CLR. Also, install the .NET CLI tooling. The required links are listed at the end of this article. Start a command prompt in your environment. This might be a command or terminal window depending on your operating sys-

tem. Create a new folder on the file system named HelloWorld and enter that folder from the console. Run the command `dotnet new` and inspect the generated contents.

project.json:

```
{
  "version": "1.0.0-*",
  "compilationOptions": {
    "emitEntryPoint": true
  },

  "dependencies": {
    "NETStandard.Library": "1.0.0-rc2-23616"
  },

  "frameworks": {
    "dnxcore50": { }
  }
}
```

The frameworks section of the project.json file lists all the targets for which this project will be compiled. It targets a single framework `dnxcore50`, which is the TFM for .NET Core 1.0 (and might change in the future). Since this project is a console application and not a library, it targets the .NET Core Platform specifically, not one of the `netstandard` targets. It depends on `NETStandard.Library`, which is a placeholder that bundles the dependencies for platform agnostic assemblies (e.g. `System.Linq`, `System.Runtime.Numerics`) and the `NETStandard.Platform`. In its turn the latter is a placeholder package that contains contract assemblies representing the surface API of the new .NET dependencies on the base class libraries in the Platform Standard. The hierarchy of dependencies is shown below.

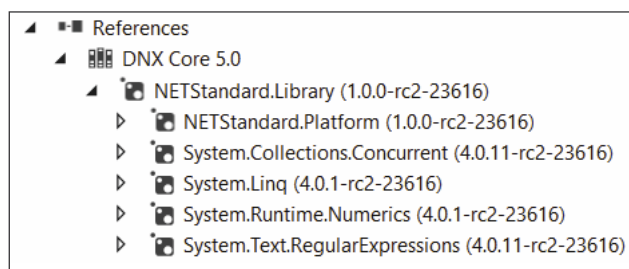


Figure 4

The code for our skeleton application is as follows:

Program.cs

```
using System;

namespace ConsoleApplication
{
    public class Program
    {
        public static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```

The skeleton program is exactly the same as for .NET 1.0 back in 2002 since the basics of .NET have not changed much. It is only when you start using more recent libraries that you will encounter the differences. This is most noticeable when creating ASP.NET Core 1.0 applications, as it is targeted at `netstandard1.4`. Referring to the matrix with the `netstandards`, you can tell that libraries from ASP.NET Core 1.0 can only be run on full .NET Framework 4.6.x and .NET Core 1.0 (referred to in the picture as DNX Core 5.0). These require the specific runtimes CLR and CoreCLR respectively, where CoreCLR is the only one that has cross-platform implementations for Linux, OS X and FreeBSD.

Now, run the commands

```
dotnet restore
```

```
dotnet build
```

This will restore the dependencies and build the code against the packages that will be downloaded from the specified NuGet feeds. The NuGet feeds are specified in the `NuGet.Config` file that was also generated by the .NET Initializer tool (`dotnet new`). Most notable is the inclusion of the feed for .NET Core itself at <https://www.myget.org/F/dotnet-core/api/v3/index.json>.

Assuming the restore and build are successful, you should be able to run the `HelloWorld.exe` executable by issuing the following command from the project folder:

```
dotnet run
```

This will only display the infamous words "Hello World!". Although not very impressive by output it is rather remarkable that this flow will work regardless of the operating system you were working on. Your executable was compiled targeted at dnxcore50 (i.e. .NET Core 1.0). .NET Core has multiple CoreCLR runtimes for the various OSes and can even use the .NET CLR runtime as an alternative on Windows.

Managing runtime environments

The final piece of the puzzle of .NET Core is the management of the various execution environments that have the runtimes. The .NET execution environments (DNX) are versioned and targeted at an operating system, processor architecture and runtime. The .NET Version Management tool (DNVM) manages the environments that are installed on a machine. Try running the tool from a command prompt to print a list of the installed DNX versions:

```
dnvm list
```

The output is similar to this, but will vary based on what was previously installed.

```
C:\>dnvm list
Active Version      Runtime Architecture OperatingSystem Alias
-----
1.0.0-rc1-final     coreclr x64      darwin      macosx
1.0.0-rc1-final     coreclr x64      linux       linux
1.0.0-rc1-final     coreclr x64      win
1.0.0-rc1-final     coreclr x86      win
1.0.0-rc1-update1   clr      x64      win
1.0.0-rc1-final     clr      x64      win
1.0.0-rc1-final     clr      x86      win
* 1.0.0-rc2-16357    clr      x86      win      default
```

Figure 5

The tool also allows you to choose the active DNX that determines the actual runtime (CLR or CoreCLR) for executing your application. You can change the active DNX by commands like

```
dnvm use 1.0.0-rc1-final -r coreclr -a X64
```

where you specify the version, runtime and architecture of the processor. This must be one that can run on the current operating system if you want to execute your application. However, you are free to install and package runtimes of other operating systems to prepare deployments of the application to those OSes.

Conclusion

.NET Core is the latest addition to the existing set of .NET Platforms. It is cross-platform and open source and a modernized implementation that enables scenarios such as containerization of the applications built with it. Although much is still the same, there are substantial differences under the covers, including the existence of multiple runtimes, compilers and tooling that come with .NET Core. The future of .NET is looking bright and .NET Core is a first impression of what's in store. The way .NET Core is designed allows a quick evolution of the .NET Framework in new directions and across platforms never deemed possible before.

Links

- Install .NET Core:
<http://bit.ly/coreclr>
- .NET CLI tooling:
<http://bit.ly/dotnetcli>
- .NET Platform Standard:
<http://bit.ly/netplatformstd>



ALEX THISSEN

CLOUD LEAD CONSULTANT
XPIRIT



Alex assists companies in building web applications and back-end solutions using Microsoft technologies and frameworks. He helps with the migration of existing architectures to modern standards and designs, as well as cloud solutions running on platforms such as Azure. Alex cares about security and informs organizations and development teams about secure coding and best practices.

How Docker will change Microsoft development

Within the developer community, Microsoft has always been known for their great IDE, Visual Studio, and their development framework .NET. Together with products such as SQL Server, SharePoint and BizTalk, this was the basic set of tools of every Microsoft Developer. In addition to the Microsoft toolset there were some additions such as HTML, JavaScript, CSS or some other third-party frameworks and tools, but Microsoft's portfolio constituted the main body of the tooling required by developers.

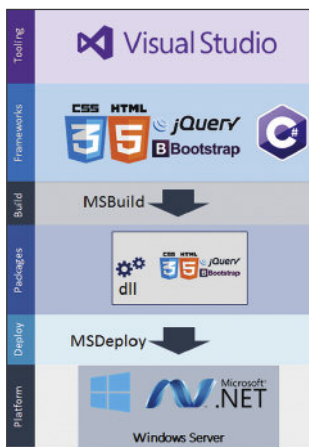


Figure 1:
The 'old' Microsoft stack

The downside of this unified toolset and the tight coupling with Windows caused more and more developers and companies to shift away from the Microsoft platform and for various reasons. Companies did not want vendor lock-ins or to rely solely on proprietary software. Moreover, developers wanted more choice and control over the frameworks they used, preferring to move to Open Source.

Microsoft created a lightweight version of the .NET runtime called the CoreCLR. This CoreCLR is available on all the platforms and is much lighter than the traditional CLR. And although it seems trivial because most of the ASP.NET applications run on Windows this actually opens up a whole new range of possibilities.

ASP.NET 5 is a powerful technology. However, the downside always was that it was not Open Source. But now that it is Open Source, ASP.NET has the power and curation of a large company like Microsoft but all the benefits and community of Open Source, allowing the technology to evolve even quicker.

With the CoreCLR, which is an important part of the ASP.NET stack, every developer is able to build and run applications on all platforms.

From mainframes to containers

With regard to application development and deployment, a lot of attention is currently paid to Continuous Delivery and DevOps, i.e. the ability to release an application on demand, and the ability to remove all friction between the people building the application and the people running it.

In a "mobile first, cloud first world", where computing power is ubiquitous, availability and performance is the most important thing and where complexity is increasing rapidly, we see that the industry is ready for the next step. What is needed is higher density, less downtime, faster start-up times and immutable software.

When we started out in IT, we used mainframes. Because mainframes were not flexible, not accessible to everybody and were expensive to run, physical servers became popular. After that Virtual Machines were introduced, because physical servers did

Times are changing

But Microsoft has changed its strategy – it is moving to a "mobile first, cloud first" world and knows that there is more than Microsoft alone. And this shift of strategy also impacts the Microsoft Developer.

Some major changes were made to overcome the difficulties faced by companies and developers using the Microsoft platform, there were. Microsoft does not only want to offer the best platform and IDE for Microsoft developers, it also wants to offer the best platform for all developers, regardless of the platform or technology being used. We now see Microsoft adopting Open Source, providing development tools on every Operating System (Visual Studio Code) and it is even open-sourcing its own technology. The .NET platform became Open Source and ASP.NET 5 was completely rebuilt to be able to run on both Linux and Windows.

not scale very well, were hard to clone or move, and also involved considerable operational cost.

And to take this a step further, the Virtual Machines moved towards a Cloud Platform to further reduce costs and increase the scalability and reliability.

And now it is time for the next step in computing: Containers. Virtual Machines have a large footprint, are hard to maintain because they all have their own Operating System and configuration and they are quite heavy to run, allowing only a few Virtual Machines on one physical host. To overcome this challenge, containers are the ideal solution. They offer the benefits of a Virtual Machine but without the related overhead and footprint.

So when Docker was introduced a few years ago, it built on the already existing container technology in Linux, and containers really took flight. Because Container Technology is the next step in computing and because containers are currently only possible on Linux, it is not surprising that Microsoft is working hard to support containers on the Microsoft platform as well. And this is what will happen with Windows Server 2016 and Windows Nano Server.

Docker, Images and Containers

Before we discuss the possibilities that containers can offer the Microsoft developer, let's take a look at what container technology is actually all about. What makes it different to Virtual machines? It is important to understand the concepts behind a container, especially because a single container cannot run on both Windows and Linux. In other words, container technology in itself does not offer OS transparency.

As illustrated in Figure 2, we can see how Virtual Machines work conceptually. An Operating System is installed on the physical Hardware, and by using a Hypervisor, Virtual Machines are created. They run on virtualized hardware and are assigned resources by the Hypervisor. We can run multiple Virtual Machines on one host and they are totally independent of both the host and one another. One Virtual Machine can run Windows 2012, one can run Windows 2003 and one can run Linux. Applications are installed on the Virtual Machine. As far as the application knows, it runs on physical hardware. This also applies to the users and administrators.



Figure 3: Containers

This independence of the operating system within a Virtual Machine is perhaps the largest difference between Virtual Machines and Containers.

As illustrated in Figure 3, containers run directly on the Operating System, without using a Hypervisor or virtualization layer. The container uses the underlying Operating System (or actually Kernel) and only saves the delta as part of the container. For example, actions such as installing software, changing settings or creating files are all stored as a new layer on top of the base container image, which is the same as the underlying OS. This means that a container is much more lightweight than a Virtual Machine because we don't have the footprint of the full OS. This allows us to run more containers on one host and be much faster in startup time. Moreover, we keep the benefits of virtualization because the containers are also independent of each other.

As you can see in Figure 4, containers are layered. You start out with a base image, which is the "transparent layer" on top of the underlying Operating System. In case of Windows containers this is a Windows Server Core image or a Nano server image. All actions you perform within the container, for example installing Git or installing ASP.NET, are saved in a separate layer on top of

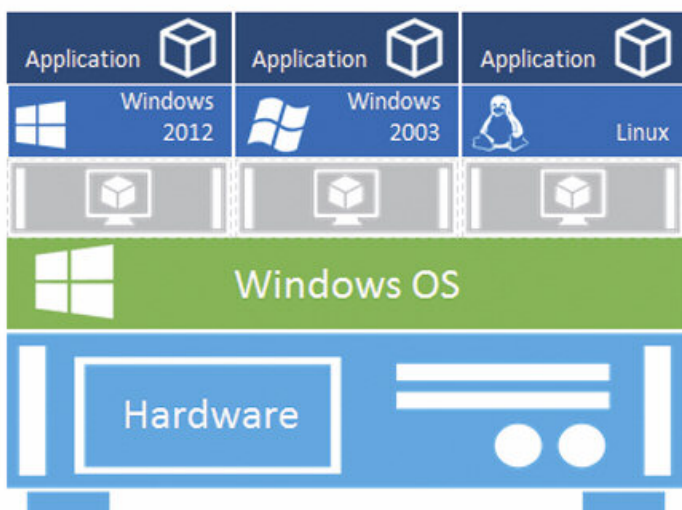


Figure 2: Virtual Machines

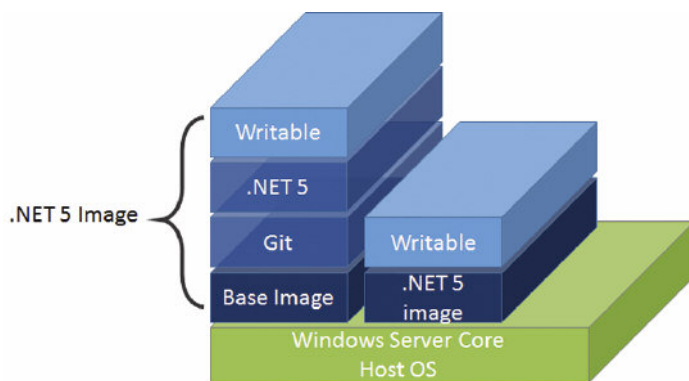


Figure 4: Containers are layers

the base image. The container including the layers can then be stored as a new image and be reused as a new base image. This way you can create an ASP.NET 5 base image that contains all necessary files, and use that image to run your application on. This makes it super easy to reuse images.

You can describe an image in a Docker file. This file describes which base image you should use and what additional actions you want to perform on this image. A Docker file could look something like this.

```
1 FROM windowsservercore
2
3 ENV vso_username="user"
4 ENV vso_password="password"
5
6 RUN install-package Git
7
8 ADD . /app
9 WORKDIR /app
10
11 ENTRYPOINT ["/kestrel"]
```

Figure 5: Example Docker file

In this example, we take the windowsservercore base image, add two environment variables, run a PowerShell Command to install Git, add some files to a folder, navigate to the folder and run the webserver kestrel.

This file describes the image of the container and can be stored in source control, be stored in a public or private registry so it can be distributed, and allows you to create new container images. You can imagine the power of this approach.

Hyper-V Containers

One of the small downsides of containers running on the same host (both on Windows and Linux), is that they are not completely isolated. This possibly involves two problems¹:

- The kernel is shared between the containers. In a single tenant environment where applications can be trusted this is not a problem but in a multi-tenant environment a bad tenant may try to use the shared kernel to attack other containers.
- There is a dependency on the host OS version and even patch level which may cause problems if a patch is deployed to the host which then breaks the application.

This is where Hyper-V containers may be the game changer for containers on the Windows platform. With Hyper-V containers a mini Hyper-V VM is spun up, which is completely isolated from other VM's and the host kernel. Inside the mini VM, the container is still used, but without the possible downsides. The only downside you have is that you have a slightly larger footprint of the larger container (or VM) and thus a lower density on the server.

On Windows it will be possible to run your container as a "normal" container or a Hyper-V Container. You can choose this during deployment time, making this a real advantage.

The impact for the Microsoft Developer

Now that we know a bit more about the inner workings of a container, and how it differs from a Virtual Machine, we can explore some new possibilities that containers can provide to the developer community.

A change in the developer experience

As a developer you use a lot of different tools, a lot of different configurations and also a lot of different versions of tools. However, managing all these configurations, or developing for a specific target requires a lot of effort. What we usually do when we want to run multiple versions of a tool in parallel is to install a Virtual Machine. We can now run a VM quite easily in the cloud, but we can also use a Virtual Machine on our own Hyper-V server. Regardless of the option we choose, it requires a lot of resources. With Docker containers, we can run lightweight containers, with their own tools and configurations and versions, all on the same host. And we can run many of them at the same time. But the best part is, we can also throw them away and keep the configuration of our development/test environment in a file

¹ <http://windowsitpro.com/windows-server-2016/differences-between-windows-containers-and-hyper-v-containers-windows-server-201>

instead of a large Virtual Hard Disk. So instead of only having the code for a specific version of our application, we also have the development environment. When we need it, we create a new image and we run it in a matter of seconds. Of course, we can also use containers to facilitate our dev/test experience on the local workstation. Currently, Docker on Windows is in its early steps of development and the developer tools are not yet adjusted to conveniently work with containers. Besides the Docker plugin² in Visual Studio, which allows you to directly publish an ASP.NET 5 website to a Docker container³ and the Yeoman generator⁴ to scaffold a Dockerfile for your application, there are no tools available yet. But imagine the power of a container that can be used as a development tool. You build your code and publish it to a Docker container and spin it up. Once running, you can debug your code within the container. No caching trouble, or files that were still installed or version-specific issues. You can create images of containers that represent the state of the production environment that you are developing for. And the best thing is, the container definition becomes part of your source code because you can store it along with your code in source control.

A change in the Continuous Delivery process

The most obvious change containers will bring, is a shift in how we build, test and deploy our applications. One of the biggest advantages of a container is that it is immutable. You create the image, which describes everything that should be in the container and configure its properties. The application is deployed within the container and configured as well, and this immutable container moves through the various stages of Test, Staging and Production. This is a dream in a DevOps world where we do not want big documents and configuration settings being handed off to someone who can do this on a production machine.

When containers are used for moving an application through the various stages, it also means that automation becomes even more important than it already is. It means that the creation of the container image, the building of the application and the configuration all need to be done automatically. Build Servers, Release Management and Configuration as Code become crucial. Testing the application becomes easier as well. Instead of deploying to one Test Environment, each version of an application can be spun up as an extra Test Environment, allowing testers to have fine-grained control over the bits they are testing.

The following figure gives an idea of the structure of a typical delivery pipeline:

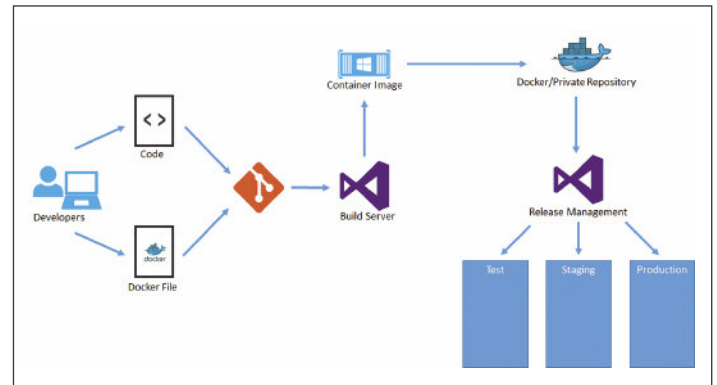


Figure 6: Continuous Delivery workflow

The figure shows that a developer pushes code and a Dockerfile to the source code repository. The build server picks up the latest version and builds a container image from the source code and Dockerfile. The container contains the application and configuration, and is sent to the Docker repository (DockerHub) or a private Docker repository. Such a repository keeps a copy of Docker images that you can pick up and reuse.

A typical example could be the creation of a base container image that contains all the prerequisites, and then adding the compiled sources to this container. From here the container images can be pulled by developers but also by Visual Studio Release Management (or a build server etc.). VS Release Management installs (or spins up) the containers in the Test, Staging and Production environments.

A change in application architectures

When you run lightweight containers that are immutable and disposable, it makes no sense to run a big monolith inside the containers. The results will become visible in the future, but the fact is that using containers and using a loosely coupled, microservices-like architecture are related to each other. We could start using containers for our microservice architecture, or perhaps we will move to microservices because we want to use containers. Whatever the results may be, containers will play a major role in how we architect and build applications. Because when we have a disposable resource that we can recreate by using a file and spin up new instances in a matter of seconds, it makes no sense to architect our application for updates. We always create a new instance, and we can use this to our advantage. We can create a new version of a container and spin it up next to an old one. Services that use the old container can start pointing at the new

² <http://aka.ms/dockertoolsdocs>

³ <https://visualstudiogallery.msdn.microsoft.com/0f5b2caa-ea00-41c8-b8a2-058c7da0b3e4>

⁴ <http://aka.ms/yodocker>

one and the old ones can be shut down. If needed, we just spin up a new instance of the container.

For example, take the following situation: imagine that you have a tax office and need to calculate tax. Every year the application is changed: new rules, new data. The calculation takes a fair amount of time, so incoming requests are stored in a queue.

In a traditional situation you would probably have a Load Balancer and distribute the requests to multiple servers. When using containers, you can use a completely different approach. Instead of one application that is updated from time to time, the processor that takes the top of the queue, gets the latest version of a container and spins it up. The container processes the request and is then thrown away.

If a newer version is released, the newer version is used instead. No downtime and no traditional problems of an application that takes a long time to run.

Summary

Containers have been around for quite a while on the Linux platform and now they are coming to the Windows platform. A container is really different from a Virtual Machine, because it is small, fast and immutable. When it comes to developing software, containers will change the way we work. Developer workflows will be optimized for building and testing in containers, the delivery

workflow will be optimized for delivering containers, and the entire application architecture will be modified to be able to deal with that.

Extra Links

- Channel 9 Videos – <http://aka.ms/dockerfordotnet>
- Getting started – <http://aka.ms/windowscontainer>



RENE VAN OSNABRUGGE

ALM LEAD CONSULTANT
XPIRIT



René is constantly promoting improvement in all areas. Using modern technology, implementing Continuous delivery and DevOps practices, and coaching on Scrum and Agile, he assists companies in improving their software delivery process. As an MVP in Visual Studio and Development Technology, René is an active blogger and speaker at both national and international conferences where he shares his knowledge of his passion: Application Lifecycle Management.

Build the future of apps
at the largest cross-platform
mobile event in the world.

Xamarin
 **EVOLVE16**
The future of apps

Hyatt Regency

April 24 – 28, 2016 Orlando, FL

evolve.xamarin.com

Scaling Scrum Professionally using Nexus and Visual Studio Team Services

If you have been using Scrum to develop products, you have probably found that the Scrum Guide only describes the core rules of Scrum, regardless of scale. A lot of companies want to use Scrum to work on multiple products or to develop a comprehensive product that requires multiple teams. The question then arises of how to organize product ownership. Even more common are situations in which the eyes are bigger than the stomach, where the list of features is much longer than their current team can deliver and where companies need to scale to multiple teams to deliver more items of the backlog within a given timeframe.

All of these cases require some form of scaling, but the element which is scaled is different in each of these situations. Yet in all of these cases, dependencies are increased, between teams, between Product Backlog items and sometimes even between different products. Managing these dependencies requires changes in the way people work (processes and people) and this is difficult without the help of supporting tools.

Many organizations have been scaling Agile and Scrum for years. And for a long time there was a list of commonly used practices and patterns used in the Agile Community, but nothing fully defined. More recently, scaling frameworks such as SAFe and LeSS have been released. SAFe appears to be a complete methodology that tries to encompass large and complex organizations from portfolio management and budget allocation all the way down to the team organization. On the other hand, LeSS mainly focuses on large development organizations with a single product or product family, while it also includes mandatory organizational changes.

Scrum.org has released its vision on scaling in the same succinct way that Scrum is defined, i.e. with the release of the Nexus Guide¹ that introduces the Nexus Framework, as well as the Scaling Professional Scrum course and assessments. The course introduces the Scaled Professional Scrum practice library. The iterative and incremental approach to scaling is what makes Nexus stand out from the alternatives. The Nexus framework strongly empha-

sizes technical excellence in order to scale Scrum sustainably, recognizing that many people who use Scrum are still having trouble delivering a Done increment of working software as one team. Obviously, things only get worse when scaling up, when more teams are involved.

Personally, I am much more comfortable with this empirical approach of technical excellence than any big-bang scaling effort which tries to change the tools, technology, teams and organization all at once.

Microsoft has developed a Scrum template in the past, and the Agile experience in Visual Studio Team Services has recently been revamped. Most of these new agile features have also shipped as part of Team Foundation Server 2015 update 1.

In this article we will focus on the way in which Nexus relates to Scrum and then look at some of the challenges experienced by teams and how tools such as Visual Studio Team Services can support them.

Introducing Nexus

In order to scale Scrum, the creators of Nexus stayed away from a prescriptive approach to scaling, because experience shows that no single solution to Scrum, let alone at scale, works in every possible environment, not even within the same organization. Over time, and with every release of the Scrum Guide since 2010, Scrum has been simplified by removing things that were too prescriptive.

¹ <https://www.scrum.org/Resources/The-Nexus-Guide>

No scaling method or process should undo those simplifications. This is why it shouldn't be a surprise that Nexus is a lot like Scrum. As Ken Schwaber, co-creator of Scrum.org describes it, Nexus uses Scrum to scale Scrum. Nexus focuses on complex product development in which multiple teams work on the same product or product family.

When working on one product with one or two teams, Scrum suffices and the overhead of a scaling framework doesn't add value, it may even do harm. When working with multiple products and multiple teams, each product can use Scrum or Nexus to manage their product development, in addition, portfolio management may be required for budgeting and prioritization across products, projects and other investments. However, Nexus does not cover this topic.

Nexus uses Scrum to scale Scrum

Balancing multiple products across a single team tends to be chaotic. As long as the work is not complex, this may be possible, but for complex work such as software development, this approach is not sustainable. Kanban can help organize the work, but only if the level of complexity can be kept to a minimum (support, operations and maintenance are examples that tend to work relatively well in this model).

The various combinations can be visualized in the following way

	 Multiple Products	 One Product
 One Team	Kaban or chaos	Scrum
 Multiple Teams	Portfolio Management with Nexus or Scrum at for each product	Nexus

Figure 1

Nexus does not attempt to resolve the challenge of an Agile organizational transformation, nor does it resolve the common situation in which one team tries to balance its capacity across multiple products at the same time. It provides a simple frame-

work in which multiple teams can learn how they work best together on one or several products.

To better understand how Nexus is an extension of Scrum, let's look at the differences and similarities between Scrum and Nexus.

Scrum	Nexus
Scrum is a Framework, and it can encompass many good agile practices. Practices that work for one product and one organization may work differently for other products and organizations.	Nexus is a Framework, and to make it work, it needs to be extended with good agile practices. Practices that work for your product and organization may be different than those of another organization. Due to the added complexity of working with multiple teams, these practices are even more important.
Scrum focuses on teams of 3-9 people. Fewer people don't need the overhead of scrum. More people cause communication to break down.	Nexus focuses on 3-9 Scrum teams, which as a whole are referred to as the Nexus. Fewer teams don't really need the overhead of a scaling framework. More teams will be unable to work together effectively.
A Scrum Team has all the skills required to deliver an increment of working software.	A Nexus consists of teams who have the skills required to deliver an increment of working software. Nexus allows for the existence of component or layer teams, but all work done in the Sprint across all teams must deliver value together.
A Scrum Team works on a single Product.	A Nexus works on a single product or product family.
A product has a single Product Owner, though some responsibilities may be delegated to the teams.	A product has a single Product Owner, though he may delegate some of his responsibilities to the teams.
A Scrum Team delivers an integrated increment of working software every Sprint	A Nexus delivers an integrated increment of working software every Sprint.

Scrum	Nexus
Sprints are 30 days or less.	Sprints are 30 days or less.
The main events of Scrum are Sprint Planning, Daily Scrum, Sprint Review and Sprint Retrospective. They are contained in a Sprint.	The main events of Scrum are Sprint Planning, Daily Scrum, Sprint Review and Sprint Retrospective. They are contained in a Sprint. The main events of Nexus are the same as those in Scrum with the addition of Product Backlog Refinement as a new time-boxed event.
	Most events are extended to handle cross-team interaction, indicated by the addition of a Nexus component. So there's a Nexus Sprint Planning, Nexus Daily Scrum etc.
The main artifacts of Scrum are the Product Backlog, Sprint Backlog and the Increment.	The main artifacts of Nexus are the same as those in Scrum, with the addition of the Nexus Sprint Backlog, which shows the dependencies between the Sprint backlogs of the individual teams.
The Product Owner, Scrum Master and Development Team are the only roles that exist in Scrum. . . .	Nexus adds a Nexus Integration Team (or NIT) which consists of the Product owner and those members of other teams required to facilitate integration.

Figure 2

Even graphically Nexus looks remarkably similar to Scrum, which makes it easy to understand for organizations that have been applying Scrum for some time now. You can see the word Nexus popping up a number of times, usually just before or just after a standard event.

NEXUS™ FRAMEWORK

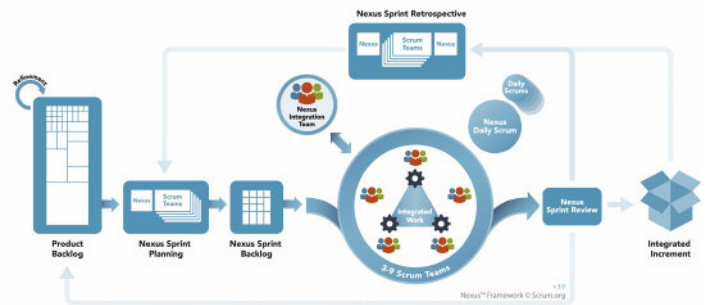


Figure 3

- **Scrum Team and Development Team** - unchanged. The Scrum Team still consists of a Product Owner, a Scrum Master and a Development Team, consisting of people with all the skills required to deliver the work required.
- **Product Backlog** - Unchanged. It is still a single ordered list of Product Backlog items owned by a single Product Owner.
- **Product Backlog Refinement** is introduced as an official event. It happens during the Sprint and there may be multiple events (sometimes in parallel) to which teams send one or more team members. It is important that these events are attended by the right representatives from the individual teams. One of the primary goals of these refinement events is to see how work can be broken down or combined to remove dependencies between teams.
- **Sprint** - Unchanged. It is still a time-box in which all the events of Scrum and Nexus take place.
- **Sprint Planning** is extended. Before teams start their Sprint, the Product Owner shares the vision and the primary goals for the next Sprint. Delegates from each team come together to discuss Product Backlog items that are mutually dependent. Afterwards, each team launches into a normal Sprint planning meeting, so all teams plan their Sprints at the same time. This works best when all teams are in the same room, although this is not mandatory. This combined event is called the **Nexus Sprint Planning**.
- Each team still has its own **Sprint Backlog** to ensure that all work is transparent within the teams. One additional artifact is created, the **Nexus Sprint Backlog**. This shows the Product Backlog items each team will work on during the Sprint. It also visualizes the dependencies between these items and teams.

- **The Nexus Daily Scrum** happens on a daily basis, and takes place before each individual team meets in their Daily Scrum. In the Nexus Daily Scrum delegates from all teams come together to flag any impediments or dependencies from other teams that will impact the plan for the next 24 hours. The primary source of inspection of a Nexus Daily Scrum is the state of integration of the work of the Nexus.
- All work that's been done, i.e. work that is fully integrated across all teams and delivered as an **integrated Increment** of product, is presented at the **Nexus Sprint Review** for feedback. All teams deliver this review together. There are no individual Sprint Review events.
- The Sprint ends with the **Nexus Retrospective**, during which all teams together discuss how the Sprint went at the Nexus level and which things may need to be improved. The outcomes are taken into the Sprint Retrospectives of the individual teams. At the end all teams come together to share changes to their process, team or tools, and share anything that might be useful for the other teams.
- Nexus introduces one new "virtual" team, the **Nexus Integration Team**. Its role in the Nexus is to facilitate collaboration and integration across the teams, and remove impediments at the Nexus or organizational level. The team is made up of the Product Owner and the "right selection" of team members from all Scrum Teams. Its composition may change over time, as the problems they try to solve changes.

Nexus assumes that the Scrum Teams will know how to solve the problems they are facing and how to best implement the work on the Product Backlog. It does not

Scaled Professional Scrum is a library of practices that have been proven in the field in different situations provide prescriptive guidance on how to conduct a Sprint Planning meeting with 9 teams, nor does it specify how to remove dependencies between teams. This is where Scaled Professional

Scrum provides building blocks to extend the Scrum framework to the level of 3-9 Scrum Teams developing and sustaining one product.

Introducing Scaled Professional Scrum

The main reason why Nexus can be explained in only 10 pages is also the reason why Scrum.org is developing Scaled Professional Scrum (SPS). Nexus provides a framework that shows how to work together and how to improve incrementally. Scaled Professional

Scrum is a library of practices that have been proven in the field in different situations. It also includes practical guidance and a two-day workshop to gain hands-on experience. You may compare it to the good old Gang-of-Four Patterns and Practices book. If you use Nexus and Scaled Professional Scrum, you will discover what is the best way to scale Scrum for your organization.

You can test your knowledge of Nexus, Scrum and common scaling practices by taking the Free Nexus Open assessment. Certification can be achieved by passing the Scaled Professional Scrum assessment.

Combining Nexus with the practices from SPS and tools from Visual Studio Team Services

We have observed how even one Scrum Team often has a hard time producing a releasable increment at the end of each Sprint. This becomes exponentially more difficult when you are integrating and testing work that needs to happen across multiple teams, at least every month, or less frequently.

Most agile practitioners, coaches and trainers will swear by physical boards, and actually holding sticky notes in their hands. Indeed, it changes the level of involvement of people, and it may give them a feeling of accomplishment when physically moving a task or PBI to Done. Yet when working with nine teams, especially if they're not all in the same location, it is unlikely that teams can be effective with physical boards only. It has become common to work together remotely, in different time zones or have team members working part-time for the company, tools and technology become extremely important for providing the required support to deal with these collaboration issues.

While the physical world works best to facilitate communication, collaboration and knowledge sharing, technical practices such as Source Control, Continuous Integration and automated testing help teams to work together and integrate at a technical level. Any team who has used these tools is unlikely to go back.

Let's take some of the most common scaling challenges and look at how people, processes and technical practices can help us 'resolve or reduce these.

■ Coordinating dependencies

The first way to resolve dependencies between teams is to proactively identify them, and try to work around them. This is why Nexus introduces Product Backlog Refinement as an official

event. When working with multiple teams, Refinement is no longer an optional activity. Instead, it becomes a crucial forward-planning event. Refinement is not about writing User Stories and acceptance criteria; it is about figuring out which dependencies exist in the current Product Backlog and how to reduce or completely remove the dependencies. The means to do so may include knowledge exchange, shared code ownership, rewriting Product Backlog items or decomposing or splitting functionality.

To reduce dependencies between teams and increase their autonomy, it helps to have teams that are able to work independently in defined areas of the product you're building. Cross-functional teams centered around one or multiple related feature areas are strongly recommended as opposed to component or layer teams.

However, some coordination across teams will always be required, and the Nexus Sprint Backlog makes these dependencies transparent. The teams share useful updates on integration and other dependencies in the Nexus Daily Scrum, prior to the individual teams' Daily Scrum.

At a higher level, the Product Backlog is an ordered list of items. To ensure that the Product Backlog does not become fragmented, it is important to assign work to teams only just before or at the start of each Sprint. If you don't, you will end up with one Product Backlog for future work and a Product Backlog for the work each team has claimed.

This is detrimental to transparency. It will be very hard to see how work on one team's backlog relates to that of another team, and it may be that more valuable work is delayed when one team is delivering faster than another.

In many tools, including Visual Studio Team Services, it is easy to lose the grand overview of the backlog when work is pre-assigned to Sprints and teams. Visual Studio Team Services provides some good alternatives: it is better to rely on the Forecasting capabilities² to identify which work will be delivered in a future Sprint combined with Tagging³ so teams can pre-emptively volunteer for specific items on the Product Backlog. Visual Studio Team Services also provides integration with tools such as HipChat and Slack, which helps teams communicate quick status updates and find answers to questions from other teams more effectively. These digital channels make it easy to share links to Product Backlog Items, Build artifacts, etcetera.

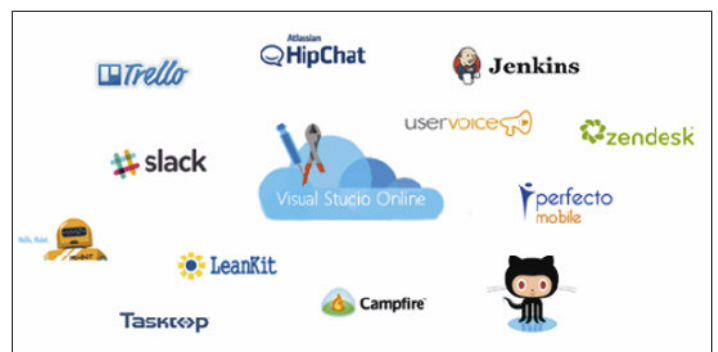


Figure 5

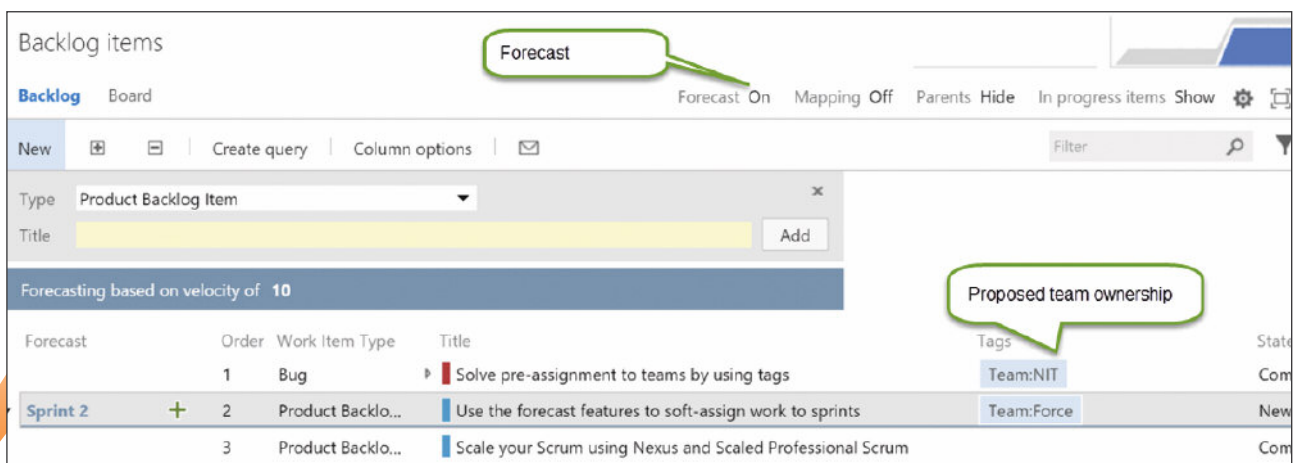


Figure 4

² <https://msdn.microsoft.com/library/vs/alm/work/scrum/velocity-and-forecasting>

³ <https://msdn.microsoft.com/en-us/Library/vs/alm/Work/track/add-tags-to-work-items>

■ Delivering a fully tested product every Sprint

When working on a large product in short Sprints it is easy to spend most of the Sprint's time on testing, or to simply not run all tests during each Sprint. Defects may sneak into areas of the product when it is least expected, causing teams to accumulate technical debt, often unknowingly. Testing is essential to deliver a fully integrated increment of working software.

It should be part of every team's definition of Done and should thus be performed every Sprint. Testing is an important strategy to identify unwanted dependencies that were not caught proactively. The only way to prevent a manual test-avalanche, and omitting important tests, is automation. Automation can be applied at multiple levels, and Continuous Integration will ensure that the sources build, while running the unit tests will ensure that no breaking changes are introduced.

Sending the Scrum Master or the team lead may not be the most effective way to smooth dependencies or resolve issues between teams

Automated deployment to test environments for automated integration and system test ensures that components will work together as expected. With complex environments, it may also be required to automate the provisioning of such environments. Visual Studio Team Services provides

Build, Test and Release Management features that can help your teams with these continuous integration problems.

■ Effective meetings

In a Nexus you'll have an absolute minimum nine people, three Development Teams, each consisting of three members, and up to nine Development Teams, each consisting of nine members (81). Six teams of six people might be an optimal size (36).

Obviously, the Scrum Masters and Product Owner need to be added to this number. Effective meetings with such large groups are impossible or at best incredibly expensive for any given organization, unless this is facilitated effectively. This is why most Nexus events are attended by representation from each team. It is extremely important that the right people are selected to represent a team. Sending the Scrum Master or the team lead may not be the most effective way to smooth dependencies or resolve issues between teams. Instead, sending the person who has the most knowledge about the subject may be required.

Some coordination around stakeholder communication and sharing of the outcomes of such meetings may be required to prevent a stakeholder from being overloaded. Tools such as Visual Studio Team Services can help capture the outcomes of such meetings as Product Backlog items and Acceptance Criteria, but it may not be enough to just capture the information in a tool. The most effective way of communication remains face-to-face.

The shared Nexus Sprint Review is hard to maintain as an effective and valuable meeting for all people attending when each team does a show-and-tell of their individual work performed during the last Sprint. Many teams have found that a "Science Fair" or "Open Space"⁴ format works well. In these cases, teams generally start with announcements and an explanation of the agenda. Each team then presents its Done functionality and receives feedback in multiple parallel slots, allowing stakeholders to attend the topics most valuable to them.

Since meetings are most effective when conducted face-to-face and visually, having an open laptop or computer screen can reduce the value of a meeting by being a distraction. Yet, not many people take the time to create sticky notes so they can use free format ways to play with backlog items and tasks. Visual Studio Team Services can still help you make your meetings more interactive and effective though. Extensions such as "Print cards" enable you to quickly transform your digital backlog into a physical one⁵, allowing teams to conduct visual meetings where they can annotate, arbitrarily group, physically split in two items on a board, unbound by process and tools.

While the above are some common challenges and practices, you may find yourself in a different situation. Don't be afraid to experiment, try out new things and continuously try to improve the way you work as an individual, a Scrum Team and a Nexus.

Not all organizations are equal and that they don't have to be in order to scale effectively

Conclusion

Nexus is an effective way to scale Scrum; it was developed by one of the original founders of Scrum. Unlike SAFe and LeSS, which enforce a lot of structure and process on organizations, it takes into account that not all organizations are equal and that they don't have to be in order to scale effectively. It introduces a

⁴ <http://openspaceworld.org/>

⁵ <https://marketplace.visualstudio.com/items/ms-devlabs.PrintCards>

number of new concepts to help teams collaborate effectively when they grow past a certain size (generally when there are three or more teams), but it tries to limit its prescriptiveness to a minimum.

"Scrum is simple, but hard" is an often-heard phrase. Nexus isn't any different. While the framework is remarkably simple, it's very hard to carry out complex, creative software development with multiple teams and deliver new, releasable software every 30 days or less.

With technical excellence, a focus on continuous improvement and hard work, it is possible though. You will find some great examples to try in the SPS curriculum and there are boundless other great resources for ideas. Nexus as a framework allows me extend it by borrowing some practices from comprehensive methodologies such as SAFe and try them to see how they can be adapted to resolve a team's challenge.

Tools such as Visual Studio Team Services provide a complete environment to support one or more teams to work together effectively. They cover a broad spectrum from work management through to source control all the way through to building and releasing your software to production. Configuring them correctly

and using the right features to accomplish the job is paramount to getting the best value. With a little guidance and an empirical approach to applying the tools and practices, amazing results are possible.

A simple framework with the support of these comprehensive tools can help you effectively scale your product development from a single scrum team to nine teams, and if need be work around the hurdles caused by distribution across time and place.



JESSE HOUWING

LEAD CONSULTANT &
SCRUM TRAINER
XPIRIT



Jesse Houwing is a Professional Scrum Trainer with Scrum.org and a Microsoft ALM MVP. Together with his colleagues at Xpirit he helps teams improve their process, skills, tools and environment to work together effectively. Scrum and Visual Studio Team Services are his tools of choice.



Xebia university

'Professional development is a continuous learning process'

For those who want in-depth knowledge, Xpirit also offers a wide variety of trainings, in close cooperation with our colleagues from Xebia. Also in-company courses or customized/tailored courses are possible, <https://xpirit.com/training> for more details and quotes. If you want to know more about our trainings and workshops, don't hesitate to <https://xpirit.com/contact/> and we will be happy to answer your questions.



R. Cornelissen

€1250

**Native Mobile Apps Using C#
and Xamarin Across All Platforms**

<http://bit.ly/NativeMobileCsharp>

Mar 7, 2016

📍 Xebia Office, Hilversum



L. Duys

€1250

Azure - Beyond Cloud Services

<http://bit.ly/BeyondCloudServices>

Mar 21, 2016

📍 Xebia Office, Hilversum



A. Thissen

€1250

**Building micro services architecture
with Service Fabric**

<http://bit.ly/MSAServiceFabric>

May 9, 2016

📍 Amsterdam



R. van Osnabrugge

€1250

**Continuous Delivery 3.0
on the Microsoft stack**

<http://bit.ly/CDOnMicrosoftStack>

Oct 12, 2016

📍 Xebia Office, Hilversum

Using the Actor Model to create distributed applications with Akka.NET

Akka.NET is an Actor Model framework for building highly concurrent, distributed, fault-tolerant and event-driven applications on .NET. This article explains what Akka.NET is, and when and how to use it. It also describes how to get started with Akka.NET, using a simple example to guide you through the possibilities.



Why concurrent systems

Just imagine you are ordering a menu at a fast-food restaurant; the employee would need to carry out the following steps:



Figure 1. Steps to carry out by an employee

*In the Netherlands we actually order mayonnaise with our menu, see also Pulp Fiction on YouTube: <http://tinyurl.com/qjzk7g6> 1:30) The employee will carry out all tasks sequentially. However, if there is only one employee, he cannot tap the milkshake at the same time as when he is frying a hamburger, and this will result in extra waiting time for the customer. But the employee doesn't have to do all these things himself in the fast-food restaurant. Usually, multiple employees are performing tasks at the same time to give you your menu as quickly as possible. Now imagine, the fast-food restaurant is a CPU and the employee is a thread. You do not want all tasks running on one thread. But running programs in multiple threads comes with a price. There are challenges to overcome. One of the hardest things to handle in a multi-threaded world is the mutable state. In the example, six tasks are changing data at the same time. In order to handle this, you will need to use read-and-write locks to prevent multiple threads from mutating the data at the same time. A lot of code is required to handle this and still it's hard to get it right, to make it perform properly and bug-free. Consider deadlocks for example. The Actor Model is an approach that hides these hard parts for you.

Actor Model

The Actor Model uses Actors to create concurrent applications. When you apply the Actor Model you will not be bothered with complex programming concepts like concurrency and parallel programming. The Actor Model hides threads and locking for you, allowing you to focus on

business logic. The business logic can be written in light-weight classes – called Actors – that are only responsible for a single task. A popular framework that implements this model is called Akka.NET. The Actor Model that Akka.NET offers is called The ActorSystem.

ActorSystem

Console applications

In this example we use Console applications to host our Actor System. For production use we recommend Topshelf – Topshelf is a framework to create a simple console application that can be installed as a Windows service.



The class in Akka.NET that represents the Actor Model is called ActorSystem. All Actors are created by an ActorSystem and live within the context of that ActorSystem. Let's see this ActorSystem in action.

We will create a very simple application that shows you how to use Akka.NET. If you follow the steps in the article you will end up with a simplified, distributed Fastfood restaurant application. This first version will be a Console application using Akka.NET that introduces you to the ActorSystem, Actors and messaging. Later on this will be changed to a distributed application.

Open Visual Studio, create a Console Application called Client, and create a class library called Shared. For now the Shared project is optional, but it is required as a preparation for later steps in this article. Add a reference from Client to Shared. Add the latest version of Nuget package Akka to both projects. Open Program.cs in Client and type:

```
static void Main(string[] args)
{
    using (ActorSystem actorSystem = ActorSystem.Create("FastFoodRestaurant"))
    {}
}
```

The ActorSystem always has a name and implements IDisposable, which is the reason the using statement is used. You are now ready to create an Actor. Actors may sound like a new term, but actually it's not new at all. The term Actor has been around for a long time.

History



Origins of the name Akka

Akka (pronounced: Áhká)

The name comes from the goddess in the Sami (native Swedish) mythology who symbolized wisdom and beauty in the world. It is also the name of a beautiful mountain in Laponia in the north of Sweden

Back in 1973, Carl Hewitt wrote a paper in which the concept of Actors was introduced. The Actor Model was implemented in the Erlang programming language and runtime system in 1987. Erlang offered high reliability and the use of many processors without the need to have explicit code. Jonas Bonér created Akka in 2009 to bring the capabilities of Erlang to Scala and Java. Since then, Akka has become the defacto standard for building distributed solutions with the functional programming language Scala.

Akka.NET was created by Roger Johansson and Aaron Stannard and is a port of Akka for .NET projects on Windows and Mono supporting both C# and F#. Both Akka and Akka.NET are open-

source. Version 1 of Akka.NET was released in April 2015.

Akka.NET is not the only Actor Model framework for .NET. Microsoft Research designed and created Orleans. The first version was released in February 2015. This project became known through the use of the cloud services for Halo 4.

Later this year Microsoft will release Azure Service Fabric; the preview can currently be downloaded. One of the programming models is also an Actor Model.

Actor

All objects in the ActorSystem are Actors. An Actor contains behavior and state.

The Actors can only communicate to other Actors using messages. Actors run independently of other actors. After receiving a message, an Actor can execute a piece of code. Like business logic, it can call a database, write to a file or change its state. In short: anything you like.

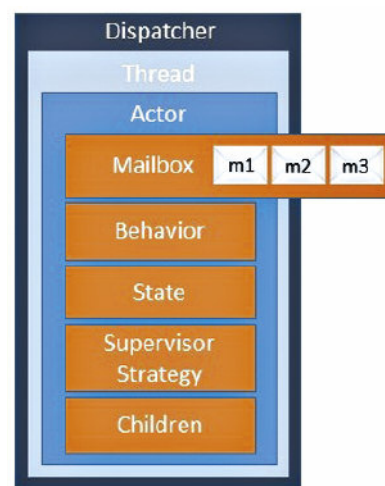
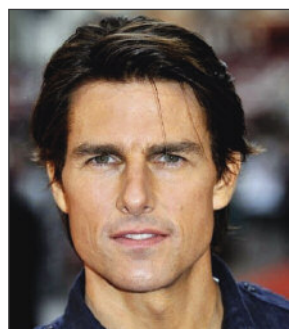


Figure 2: Actor

Besides this, Actors can create other Actors. An Actor is single-threaded and handles one message at a time.

Actors do not have a unique thread of their own. If this were the case, you would run out of resources very soon. A thread can run many Actors. When a message is sent to an Actor, the message is stored in a queue. This queue is called the Mailbox. The Dispatcher which handles the messages will notice that there is work for an Actor because the mailbox is full.



There are Actors and Actors

This Dispatcher starts a thread, brings the Actor to life on this thread, and then delivers the message to it. Perhaps you are thinking about your own domain now and trying to visualize it in Actors and coming up with a couple or even dozens of Actors.

In practice Akka.NET can handle millions of Actor instances! This changes the game of developing concurrent software and is one of the reasons why Akka is very suitable for the Internet of Things (IoT). IoT is introducing a new order of complexity for back-end systems because of the number of things, which Akka can handle by having an Actor for every "thing".

Let's define an Actor in project Shared, which can receive two types of messages. A `BurgerMenuRequest` and a `SaladRequest`. The requests, both messages, are just classes without logic or state.

There are multiple ways to define an Actor: by means of the `UntypedActor`, the `ReceiveActor` or the `TypedActor`. Each will be explained in the following paragraphs:

UntypedActor

When you extend the `UntypedActor` it needs to implement the `OnReceive` method which takes an untyped message. This is convenient when you want to receive any message. This version makes it possible to use pattern matching on messages which can decide to process the message or not.

```
public class Employee : UntypedActor
{
    protected override void OnReceive(object message)
    {
        if (message is BurgerMenuRequest)
        {
            Console.WriteLine("Collect menu");
            return;
        }

        if (message is SaladRequest)
        {
            Console.WriteLine("Collect salad");
            return;
        }
    }
}
```

ReceiveActor

The `ReceiveActor` requires you to register messages in the constructor. The message is handled in two ways in the codesnippet: both inline and as a method.

```
public class Employee : ReceiveActor
{
    public Employee()
    {
        Receive<BurgerMenuRequest>(message => {
            Console.WriteLine("Collect menu");
        });
        Receive<SaladRequest>(message => {
            Handle(message);
        });
    }
}
```

```
private void Handle(SaladRequest message)
{
    Console.WriteLine("Collect salad");
}
```

TypedActor

When you use the `TypedActor`, you also need to implement the `IHandle<>` interface for each message that you want to react to. If the logic in the Actor can handle typed messages, this is the most explicit way to implement your Actor. Therefore, create the `Employee` class as follows in project Shared:

```
// In Shared
public class Employee : TypedActor, IHandle<BurgerMenuRequest>, IHandle<SaladRequest>
{
    public void Handle(BurgerMenuRequest message)
    {
        Console.WriteLine("Collect menu");
    }

    public void Handle(SaladRequest message)
    {
        Console.WriteLine("Collect salad");
    }
}
```

Let's create another Actor, which you will need in a second:

```
// In Shared
public class Customer : TypedActor, IHandle<MarkHungry>, IHandle<Bill>
{
    public void Handle(MarkHungry message)
    {
    }

    public void Handle(Bill message)
    {
    }
}
```

For the example in this article only those two Actors are needed. If you wish, you can create Actors for all actions needed for a menu.

Creation of Actors

The example of the menu involves two Actors: `Customer` and `Employee`. How do you create an instance of an Actor? Not as usual by calling the constructor. When you do this, an `ActorInitializationException` will be triggered, telling you to use the `ActorSystem` or that another component should be used to create Actors. So let's listen to that advice and let the `ActorSystem` create an instance of the `Customer` Actor.

```
using (ActorSystem actorSystem = ActorSystem.Create("FastFoodRestaurant"))
{
    IActorRef customer = actorSystem.ActorOf(Props.Create<Customer>(), "customer");
}
```

The ActorOf method requires a Props and optionally a name. The Props instance configures which Actor to create. The name should be unique within the ActorSystem. You don't get a reference to the Customer instance, the result is an IActorRef. The instance implementing the IActorRef interface is an ActorRefImpl. Actors can only communicate with messages, so we don't need a reference. The IActorRef is a proxy to the actual instance. This instance might live within the current process or may be in a process on another computer. Akka.NET hides this for you and this is called location transparency.

```
// Alternatives to create Actors
Props props1 = Props.Create(() => new Customer());
Props props2 = Props.Create<Customer>();

IActorRef customerA = actorSystem.ActorOf(props1, "customerA");
IActorRef customerB = actorSystem.ActorOf(props2);
IActorRef customerC = actorSystem.ActorOf<Customer>("customerC");
```

Communicate with Actors

First the customer has to know that he is hungry to kick off the process. So let's communicate this through messaging:

```
// Program.cs Main
IActorRef customer = actorSystem.ActorOf(Props.Create<Customer>(), "customer");
customer.Tell(new MarkHungry());
```

The Tell method does not send the message directly to the OnReceive method or Handle method of the Actor. Instead the message will be sent through a fire-and-forget mechanism to the Mailbox of the Actor. You will not receive a response. The message will be handled asynchronously from this thread.

By default, a message in Akka.NET is delivered At-Most-Once. This means there is no guaranteed delivery. Akka.Persistence is a module that offers At-Least-Once delivery semantics. However, this is not described in further detail as it is beyond the scope of this article. The Customer Actor can handle the message with the following code:

```
// Customer
public void Handle(MarkHungry message)
{
    Console.WriteLine("Customer is hungry. Let's order some food");
    IActorRef employee = Context.ActorOf<Employee>("employee");
    employee.Tell(new BurgerMenuRequest());
    Console.WriteLine("Ordered food");
}
```

The ActorSystem is not the only class that can create Actors. An Actor can create other Actors as well. This way a whole hierarchy of Actors can be created. This hierarchy is described in more detail later in this article. Every type of Actor has a Context property which can be used for this. Now that we have a proxy to the CustomerActor, we will send a BurgerMenuRequest Message. How should the Customer be notified about the finished menu when request-response messaging cannot be used? It is possible to send a message back to the caller, also known as the parent. Of course this also happens completely asynchronously using messages.

```
// Employee
public void Handle(BurgerMenuRequest message)
{
    Console.WriteLine("Collect menu");
    Context.Parent.Tell(new Bill());
}
```

The Customer will handle the Bill Message as follows:

```
// Customer
public void Handle(Bill message)
{
    Console.WriteLine("Great, menu is ready. Let's pay.");
}
```

Until now all code handles the Happy flow. But what if an exception is raised?

Fault Handling

The way Akka.NET handles exceptions, results in isolation of faults by handling the exception locally. This way the rest of the system is not bothered and continues running. Separation of concerns is applied within an Actor regarding business logic and handling exceptions.



Restart (default)	The Actor will be recreated and will process messages, resetting internal state.
Resume	The Actor will continue processing messages, keeping internal state.
Stop	The Actor will be terminated.
Escalate	The Supervisor doesn't know how to handle the exception and escalates it to its parent, also a supervisor.

Both business logic and fault recovery logic are two different flows within an Actor. The recovery-logic monitors child Actors. An Actor that monitors child Actors is called a Supervisor. An Actor doesn't try to solve its exceptions; it will just crash, also known as the Let It Crash semantic. The Supervisor decides how to handle the exception coming from a monitored child Actor and can choose one of the following Supervision directives:

Because the parent actor handles the lifetime of a child Actor, you should not mix your business logic code with other risky code such as calling a webservice. The call to the webservice should be handled by a separate Actor. When the webservice is not available, the parent will decide how the Actor should react to the exception.

Because the Supervisor can receive multiple types of exception, a Supervision strategy can handle multiple exceptions and can decide what directive to apply to the failed Actor. There are two built-in Supervision Strategies and it's also possible to create a customized strategy.

OneForOneStrategy	Stop the child actor that failed.
AllForOneStrategy	Stop all child actors of the Supervisor.

The following code snippet shows how to program a Supervisor-Strategy. Just override the SupervisorStrategy method on an Actor and return a AllForOneStrategy or OneForOneStrategy. In this case, when an ApplicationException is thrown from a child actor, the child is restarted. Any other exception will escalate the Exception to a parent Actor.

```
public class Customer : UntypedActor
{
    protected override SupervisorStrategy SupervisorStrategy()
    {
        return new OneForOneStrategy(3, TimeSpan.FromSeconds(5), ex =>
        {
            if (ex is ApplicationException)
            {
                return Directive.Restart;
            }

            return Directive.Escalate;
        });
    }
}
```

In this example, when an ApplicationException occurs, the Actor who raised the Exception is restarted. Any other exception will escalate the Exception to the parent of this Actor. If an ApplicationException occurs 3 times within 5 seconds, the Actor will be stopped.

When the Actor is restarted it will continue processing other messages in the mailbox. This means that by default, the message which was responsible for the exception will have been removed. The simplest mechanism to keep processing the offending message is to use the PreRestart to send the message to the Actor itself.

```
public class Employee : TypedActor, IHandle<BurgerMenuRequest>, IHandle<SaladRequest>
{
    protected override void PreRestart(Exception reason, object message)
    {
        // Put the message that failed, back in the mailbox.
        // It will be picked up when the actor is restarted.
        Self.Tell(message);
    }
}
```

This solution keeps trying to process the same message, which can be a good thing for transient faults such as web-service connection-faults. But if the exception is not transient, the message will be processed forever. For this reason, you have to come up with a solution for production systems. One possibility is to apply a Circuit breaker pattern like Polly.NET.

Fault handling in Akka.NET can be done with SupervisorStrategies, but what happens when the first-created Actor, a top level Actor, throws an exception? It has no parent Actor that can handle the exception. Well actually it does.

Actor Hierarchies

Because of the parent-child setup for Actors, all Actors in a system represent a hierarchy of Actors. Akka.NET contains some out-of-the-box Actors that are available when you create an ActorSystem.

The /user Actor also referred as Guardian Actor or Root Actor is the Supervisor for the top level actors. This Guardian will handle exceptions by default with a restart directive.

All Actors have a unique address. An Actor can be placed anywhere in the hierarchy. In this example, the Employee is placed below the customer. An Actor can send a message based on the address to another Actor. To send a message, both an absolute and a relative address can be used.

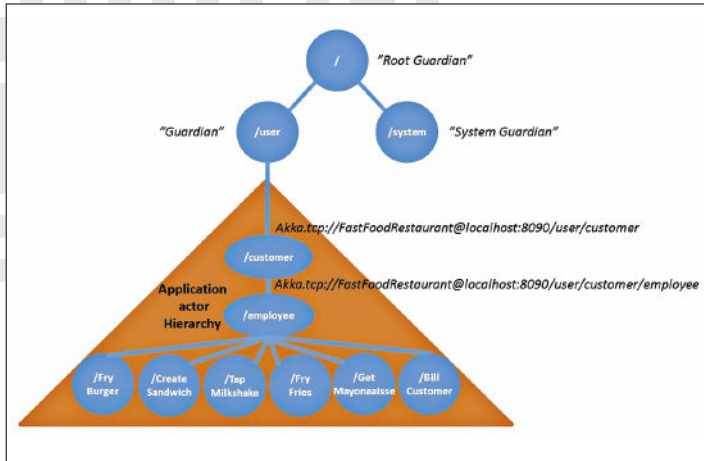


Figure 3: Actor Hierarchy

```
// absolute
var selection = Context.ActorSelection("/user/customer");
selection.Tell(new Bill());

// relative
var selection = Context.ActorSelection("../customer");
selection.Tell(new Bill());
```

The unique address of an Actor has the following structure:

Akka.tcp://remotefastfood@localhost:8090/user/customer/employee

protocol actorsystem server port guardian top-level-actor actor

Location Transparency

Until now the Actors in the example run on the same machine. When the application has reached the hardware capacity limit on the machine, you can apply a scale-out scenario. That's actually quite easy because Akka is distributed by design. Actors can run on any machine without a code change, and for the application it doesn't matter where the Actor runs. This is also called Location Transparency. Remoting is the feature in Akka.NET that offers location transparency, and it can be configured with HOCON. HOCON is the abbreviation for Human-Optimized Config Object Notation. This is the configuration format that Akka is using. This configuration can be read from a separate file or the

One Node

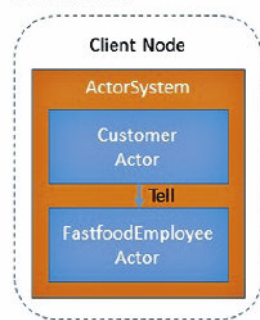


Figure 4: One Node

configuration can be embedded in the app.config or web.config inside a CData section.

To host a remote ActorSystem you first need to create a new Console application called Server. Add a reference from Server to Shared, and add Nuget package Akka.Remote to the Client and Server project in the solution. There are two possibilities for running Actors remotely: Remote Deployment of an Actor, or send a message to a remotely running Actor.

Remote Deployment of an Actor

To run an Actor on a different node (Server) it's possible to deploy it with Akka.Remote.

First we create the server. The code is the same as the code we created for the client; only the name of the ActorSystem differs. This name is important for the configuration of the client, which will follow.

Remote - Deployment

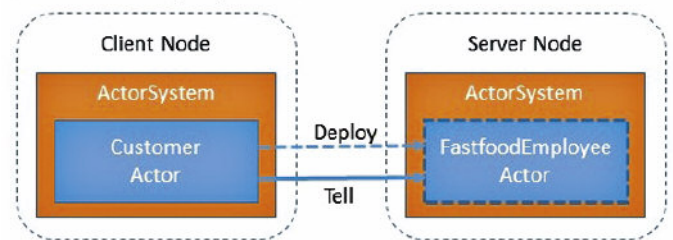


Figure 5: Remote - Deployment

```
// server
using (var actorSystem = ActorSystem.Create("remotefastfood"))
{
    Console.ReadLine();
}
```

Notice you don't initialize any Actor on the Server. The server needs an app.config file that looks like this:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <configSections>
    <section name="akka" type="Akka.Configuration.Hocon.AkkaConfigurationSection, Akka" />
  </configSections>
  <akka>
    <hocon>
      <![CDATA[
        akka {
          actor.provider = "Akka.Remote.RemoteActorRefProvider, Akka.Remote"

```

```

remote {
  helios.tcp {
    port = 8090
    hostname = localhost
  }
}
}
}
]]>

</hocon>
</akka>
</configuration>

```

The remote part tells the ActorSystem on which host and port number it is running. This is also important for the configuration of the client. The code of the client does not have to change. An Actor is deployed to a remote ActorSystem by changing the configuration in HOCON. The app.config of the client is configured like this:

```

<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <configSections>
    <section name="akka"
      type="Akka.Configuration.Hocon.AkkaConfigurationSection, Akka" />
  </configSections>
  <akka>
    <hocon>
      <![CDATA[
        akka {
          actor{
            provider = "Akka.Remote.RemoteActorRefProvider, Akka.Remote"
            deployment {
              /employeeActor/employee {
                remote = "akka.tcp://remoteFastfood@localhost:8090"
              }
            }
          }
          remote {
            helios.tcp {
              port = 0
              hostname = localhost
            }
          }
        }
      ]]>
    </hocon>
  </akka>
</configuration>

```

There are two parts in the HOCON configuration: actor and remote. Remote configures the ActorSystem on this node, the client in this case, where it is hosted. When port 0 is configured, Akka chooses a port. The actor part of the configuration configures the RemoteActorRefProvider of Akka.Remote and allows the employee Actor to run on the address of the configured server. Also the name of the ActorSystem of the Server is configured in the address. This way, when the Actor is created, it's created on the Remote ActorSystem. This is done without a code change; all actions are performed in configuration.

In this example the two host projects are called Client and Server.

Akka Remote uses peer-to-peer communication between nodes, so actually every Node is both a Client and a Server.

Send a message to a remotely running Actor

An alternative to creating an Actor on a remote ActorSystem is to host multiple ActorSystems (Client and Server) and send a message from the client to an Actor on the Server. This means that the Actors have already been deployed. To do this, first create the remotely running Actor. Create an ActorSystem like the one the Server created earlier, but with one line of code added to start the Actor to which the client will send a message.

Remote – Send message

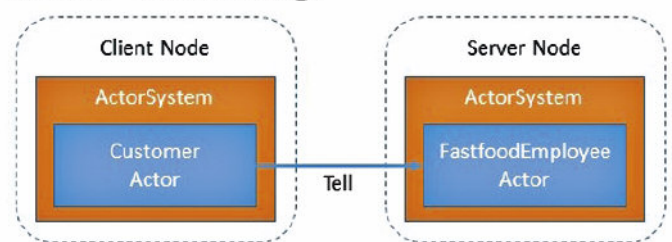


Figure 6: Remote - Send Message

```

// server
using (var actorSystem = ActorSystem.Create("remoteFastfood"))
{
    actorSystem.ActorOf<Employee>("employee");
    Console.ReadLine();
}

```

The configuration of the Server stays the same as above. The Client configuration needs a small change, but it is not necessary to remotely deploy an Actor anymore. For this reason, the deployment part is removed.

```

<![CDATA[
  akka {
    actor{
      provider = "Akka.Remote.RemoteActorRefProvider, Akka.Remote"
    }
    remote {
      helios.tcp {
        port = 0
        hostname = localhost
      }
    }
  }
] ]>

```

The code of Customer in Shared should be changed slightly. It is not necessary to create an Actor with Context.ActorOf. Instead, select an Actor based on an address with Context.ActorSelection.

After getting the proxy, you can send the message in the normal way. Because the EmployeeActor is the Top-Level-Actor in this ActorSystem, the address does not contain the customer.

```
if (message is MarkHungry)
{
    Console.WriteLine("Customer is hungry let's order some food");

    // CREATE (remote deployment)
    // IActorRef employee = Context.ActorOf<Employee>("employee");
    // GET
    var employee =
        Context.ActorSelection("akka.tcp://remotefastfood@localhost:8080/user/employee");

    fastFoodEmployee.Tell(new BurgerMenuRequest());
    Console.WriteLine("Ordered food");

    return;
}
```

The message BurgerMenuRequest is now sent to the Remote Actor. It's easy to use Akka.Remote, but the number of nodes is fixed and should be known in advance. A more advanced step is to apply Akka.Cluster.

Cluster

A cluster is a dynamic group of nodes. Just as we saw with Akka.Remote, every node represents an ActorSystem. Akka.Cluster makes it possible to create truly elastic applications by dynamically growing and shrinking the number of nodes. Cluster ensures that Actors run in a location-transparent manner; an Actor can run on any node.

Akka.Cluster makes Akka.NET highly available, fault-tolerant and ensures that there is no single point of failure for your application. Because of all these characteristics, Akka.Cluster makes it really interesting to create Microservices with Akka.NET.

To create a Cluster, you need to create nodes. It's possible to configure nodes as a Seed-Node or as a Non-Seed-Node. Seed nodes ensure that Non-Seed-Nodes can join the cluster. For this reason, Seed-nodes have a known address. The addresses of Seed-Nodes are configured at a Non-Seed-Node's configuration. Because Seed-Nodes may also fail, you should have at least two Seed-Nodes in the cluster.

Akka uses peer-to-peer communication between nodes. This means that a node is aware of all other nodes in the cluster, and communicates with them.

Akka.Cluster is not released yet. Fortunately, there is a prerelease available on Nuget.

Let's change the Client console application to create a Cluster. We will configure Client as a Seed-Node.

To create a cluster no code changes are required. You need to add a Nuget package: install-package akka.cluster-pre. The configuration should look like this:

```
akka {
  actor {
    provider = "Akka.Cluster.ClusterActorRefProvider, Akka.Cluster"
  }
  remote {
    helios.tcp {
      port = 8081 #<- configure the other Seed-Node with port 8082
      hostname = "127.0.0.1"
    }
  }
  cluster {
    seed-nodes = [ "akka.tcp://client@127.0.0.1:8081",
                  "akka.tcp://client@127.0.0.1:8082" ]
  }
}
```

It's important that you use IP-addresses or host names, but do not mix them. Also make sure that all ActorSystems have exactly the same name, otherwise the ActorSystem cannot join the cluster. If you want to run the Cluster on a single PC, just copy the bin folder of Client and change the port in the configuration.

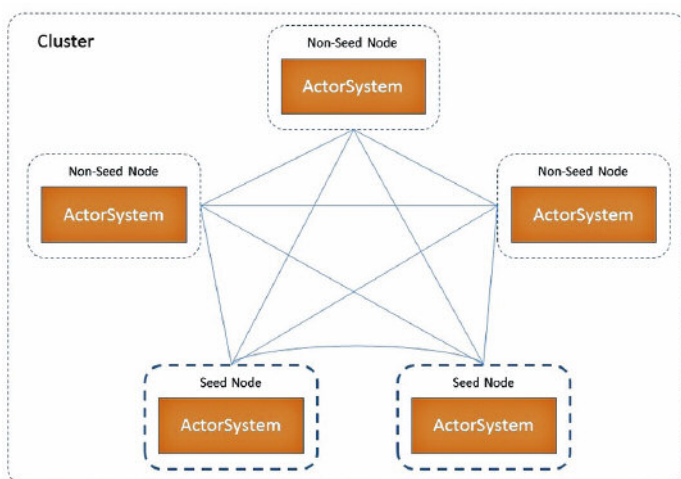


Figure 7: Cluster

Both Seed-Nodes can be started already. To create a Non-Seed-Node just copy the bin directory again and change the configuration slightly:

```
akka {
  actor{
    provider = "Akka.Cluster.ClusterActorRefProvider, Akka.Cluster"
  }
  remote {
    helios.tcp {
      port = 0 #<- no fixed portnumber
      hostname = "127.0.0.1"
    }
  }
  cluster{
    seed-nodes = [ "akka.tcp://client@127.0.0.1:8081",
                  "akka.tcp://client@127.0.0.1:8082" ]
  }
}
```

Create three Non-Seed-Nodes by copying the bin folder three times and start all clients. The logging in the Console on each node should tell you that it's connected and will start processing the message.

You have created a cluster of five running nodes now and with that you have created a Distributed application with Akka.NET that is highly available and elastic.

Conclusion

This article covers enough topics of Akka.NET to allow you to get started building applications with Akka.NET. There are many more topics to cover, for example Finite State Machines, Reliable messaging, Persistence and Routers. Not every application is suited to be programmed in an Actor-based framework like Akka.NET. If you have ever programmed multi-threaded code, Akka.NET is a blast to use. Also for applications that need millions of instances, Akka.NET gives you possibilities that are very hard without actors. The use of message-driven architectures requires a mind switch and the use and design of Actors can be challenging. Akka.NET is not as mature as Akka.

Akka, for example, provides monitoring tools and a way to create non-blocking REST-ful services with Akka.Http (Akka Spray). This way it's much easier to make the ActorSystem available through REST. Microsoft is about to release Azure Service Fabric which offers an Actor Model programming model. Akka.NET runs on Mono too, it is open-source and has a very active community. Azure Service Fabric is not available as open source yet. Because of all this, Akka.NET is a framework worth watching, and, if you so wish, to contribute to.

Sources:

- The complete source code for this article can be found on GitHub at <https://github.com/XpiritBV/XpiritMagazine/tree/master/Edition-2>
- <https://petabridge.com/blog/>
- <http://getakka.net/docs/>
- <http://akka.io/docs/>



PASCAL NABER

CLOUD CONSULTANT
XPIRIT

Pascal is a passionate software developer and technical software architect, dedicated to quality and simplicity. He has a great interest in everything related to Microsoft software development and new technologies.

Enhancing your insights with Power BI

Being informed and up-to-date on information is part of our daily routine. Just think about the number of times you reach for your phone or look at your smartwatch to see an update on headline news or a customer email. This is true in your work, but also in your personal life. If you focus on the former, you will see that there is always a need to be informed, in any organization, and especially around agile processes. This ranges from information on whether a particular feature has been released to progress of an epic that will allow the release of a new product.

The Agile Manifesto states that people and interactions should take precedence over process and tools. And it is obvious that tools will never replace human conversation and communication.

This article describes the process of enhancing reporting solutions with Team Foundation Server with the capabilities of Power BI, showing tools that can play an important role in providing adequate information.

Reporting History

Let's start by looking back at the reporting capabilities that came with Team Foundation Server (TFS). Since TFS 2005, reporting capabilities were primarily based on Microsoft SQL Server Reporting Services (SSRS) or the use of Microsoft Excel. These types of reports needed to be designed by developers who had knowledge of SSRS tools and application data, as well as the ability to translate people's questions into valuable reports. Developing these kinds of reports has proven to be quite time-consuming. Furthermore, performance has always been an issue, and browser support and compatibility has never been optimal. What's more, most of the time reports were quite static and interaction was limited to setting a large number of parameters to filter data. Changing a single parameter for filtering or refining data would require you to re-run the entire report. Figure 1 shows the parameter selection for an SSRS Report.

In addition to numerous updates of TFS, there are also updates of SQL Server Reporting Services. However, the paradigm has not changed much. Figure 2 shows RTM releases of the two products plotted on a timeline.

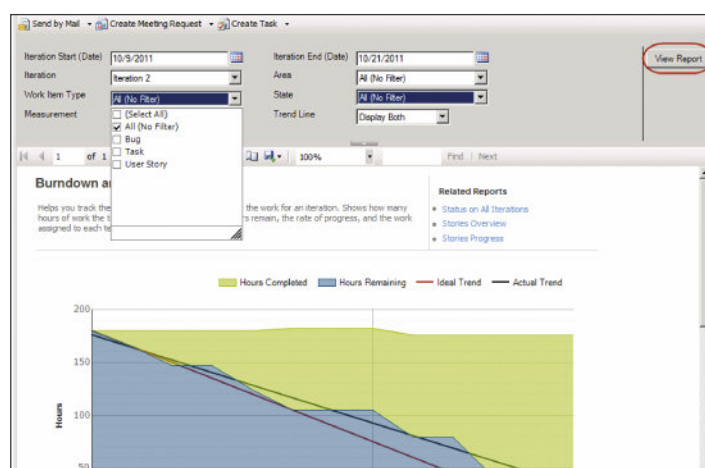


Figure 1: SSRS Report Example



Figure 2: RTM releases of TFS and SSRS

With the introduction of Microsoft Excel's 2010 Power Pivot came functionality that allows you to create reports and shape data with filters, which results in fast switching and filtering data to compare it to other selections. The use of slicers was actually a very convenient and interactive way of interacting with your data. Figure 3 shows an example of a report that combines multiple slicers to filter data.

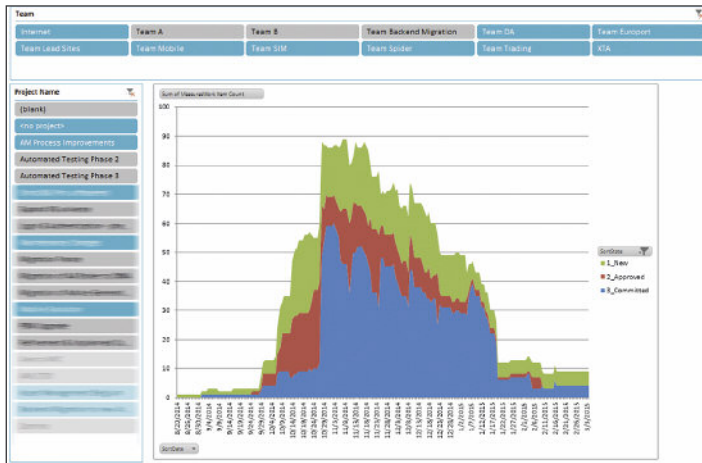


Figure 3: Power Pivot Report Example

Power Pivot makes it easy to create new reports. Moreover, overall development time is much shorter, but it turns out that this only sustainable works with a limited number of reports. Using multiple tabs with multiple charts quickly leads to large excel files, sometimes too large for excel. In that case, refreshing data can become very slow and it may sometimes lead to unrecoverable errors. Enabling scheduled refreshes is only possible by integrating them with SharePoint. However, an Enterprise SharePoint edition for publishing files or reports does not prove to be the most affordable solution.

The challenging question

Today your information needs to be updated more often. Insights need to be up-to-date when you look at them and they need to be based on today's data. Correctness and accessibility is not something you are willing to cut corners on. In addition, you want to be able to do it yourself, and get a feel of what is going on. Call it 'Do it yourself reporting' if you like. This was nearly impossible with the technologies that existed until now.

Power BI is here to change all this. Once data sources have been made available to you, you can actually create reports yourself!

Polar Express

Imagine the following scenario. You are a stakeholder for a company called "Polar Express". Your IT department consists of two development Scrum teams: Team Penguins and Team Polar Foxes. Your company objective is to be the most successful parcel shipment company in the arctic area. The teams are adding value to your company's product portfolio (Website, Mobile Apps, etcetera).

You are responsible for the IT department. Your teams are spending money fast, but when the value delivered is well received, the company's future is guaranteed to be successful.

As a stakeholder you want to be up-to-date on the progress and performance of your teams. To satisfy the informational needs and your management peers, you want to be able to monitor and respond to activities that impact the teams, without disturbing them. During a chat with one of the product owners it occurs to you that recent progress is not what it used to be. Analysis of the situation with the product owner has revealed that integration problems between the teams have led to decreased velocity.

In order to gain insight into the situation, you spoke to the product owner and told him that you would like to have detailed information on lead times as well as cycle times. However, the look on the product owner's face says it all. He does not have a report like that. A question like that has never come up before, and after a brief moment the product owner says: "I think I have a solution".

The solution

Microsoft Power BI consists of a variety of tools, a desktop application, a mobile solution as well as a web-based environment. Each of these tools allows you to connect to a multitude of data sources¹.

This article focuses on the analysis data of an on-premise TFS server. When you design reports, it is very handy to have a powerful desktop editor, along with the power provided by BI. This allows you to play around before integrating with your corporate environment. When you work with Power BI Desktop, the context for datasets, reports and dashboards can be saved into a Power BI file (.pbix). This is great for working with the environment before sharing it in the organization.

Having the ability to connect to data in Power BI allows you to create a report for it. This report can be displayed on almost any device or browser. An even better solution consists of the native support for mobile applications, which allows remote users to see and interact with reports on mobile devices. This enables users to be informed anywhere and at any time.

Shaping the data

The data for Polar Express are gathered from the on-premise TFS 2015 warehouse and analysis database. To support custom queries and custom tables, we need to use a custom database².

¹ <http://blogs.technet.com/b/dataplatforminsider/archive/2015/10/29/microsoft-business-intelligence-our-reporting-roadmap.aspx>

² <http://bit.ly/1JYIV36> blogpost by colleague Rene van Osnabrugge on creating a custom database to support custom views for reporting purposes.

In the context of this article we use a query that retrieves data from the analysis database, which is then combined with local data to produce the desired view for our report. There are many ways to connect or shape the data sources to the desired format. The primary input for the report is a view that queries the analysis server, and that will return the required data. The output of the view can be seen in figure 4.

	ActualDate	WorkitemType	States	Counts
1	11/8/2015	Product Backlog Item	Approved	5
2	11/8/2015	Product Backlog Item	Committed	7
3	11/8/2015	Product Backlog Item	Done	1
4	11/8/2015	Product Backlog Item	New	17
5	11/9/2015	Product Backlog Item	Approved	5
6	11/9/2015	Product Backlog Item	Committed	7
7	11/9/2015	Product Backlog Item	Done	1
8	11/9/2015	Product Backlog Item	New	17

Figure 4: View Results

The required source for the supporting table and views are provided in a script³ that can be downloaded from the Xpirit GitHub repository.

Once the data are available, the next step is to create a data source in Power BI that retrieves the data.

To do so, start by connecting to the SQL Server Database and then use the Navigator to select the view. Figure 5 shows the data navigator.

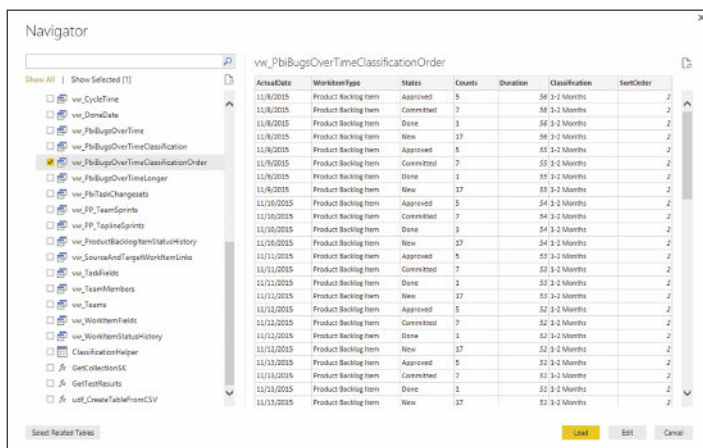


Figure 5: Data Navigator

Use the Data Tab of Power BI for Desktop to verify that our data has the correct data types. To do so, click each column header and inspect the Data Type properties.

The modeling tabs in figure 6 show several options available for data type modifications.

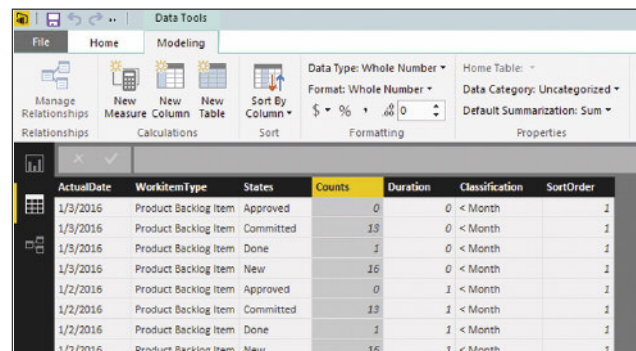


Figure 6: Data Tools, validate Data Types

Creating the report

Once the data are available in Power BI, you can start to create a report. To do so, drag a clustered column chart to the empty report. Naming the tab allows you to make a distinction between multiple reports. Set the graph to fill the page, and then configure the graph.

The right hand side shows the data available in the Fields Explorer. The left hand side shows the chart Visualizations pane. This is where you configure the chart to use the fields from the view. Figure 7 shows the section for configuring the report.

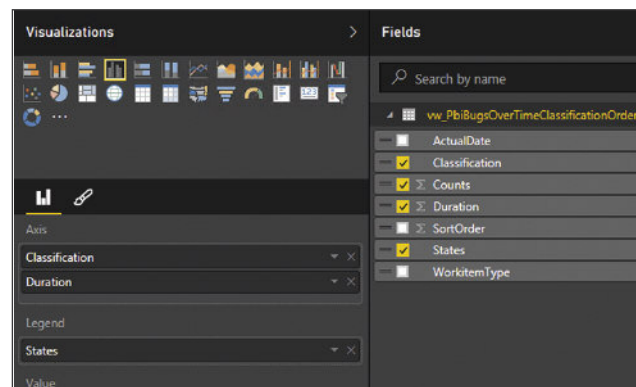


Figure 7: Configure Visualizations

Set the Axis, Legend and Value properties, and the report is immediately updated to reflect the changes. It will look similar to the report shown in figure 8. You now have a solution that you can always use in Power BI Desktop. You can save this as a .PBIX file and share this across the organization. However, that does not sound like a real improvement to previous solutions.

³ <http://bit.ly/ypowerbi>

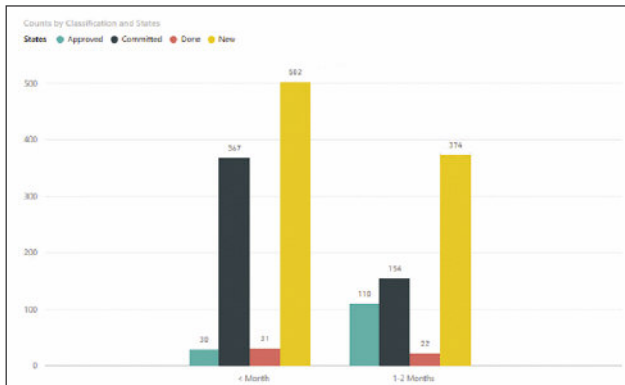


Figure 8: Report result

Publishing the report

You can extend this solution by using the Power BI Web capabilities, but you need to get a Power BI account⁴ to be able to use these capabilities.

To extend to the Power BI Web environment, you need to publish the report, which can be done by using the Publish button from the ribbon (see figure 9).

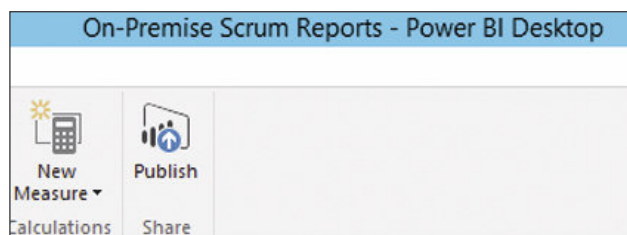


Figure 9: Publish Button

After a successful publish to your Power BI environment, the message shown in figure 10 will be displayed.



Figure 10: Publish results

Now let's go and see the Power BI Web environment for the report. Navigate to <https://powerbi.microsoft.com> and log in using your credentials.

Under Datasets and Reports you will see the On-Premise TFS item. This looks exactly the same as in Power BI for Desktop. Exposing data will be secure by using the Power BI Gateway, which will be introduced later. While reports and data are available, you want a better experience. To be able to view this report easily alongside other data, put the report on a dashboard. To do so, select the top right pin icon and pin to a new dashboard (see figure 11).

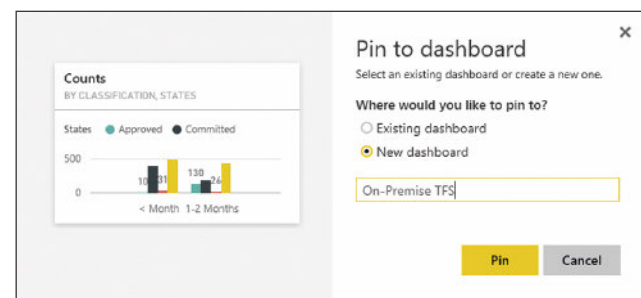


Figure 11: Pin a report to a new dashboard

Your report is now pinned to a dashboard as can be seen in figure 12.

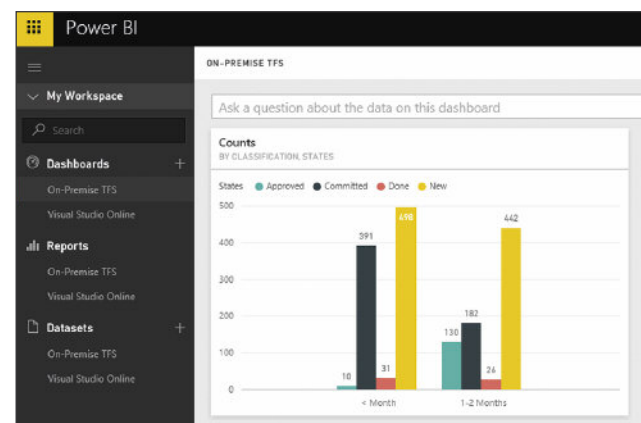


Figure 12: Dashboard overview

Mobile experience

While this is nice, you want to be able to view this on your mobile phone. In the old reporting days, this would typically mean the end of the challenge. However, Microsoft has really done a nice job by creating a native application for every available mobile platform. Although the stakeholder uses a great deal of Microsoft technology, he does run an iPhone, so he downloads the Power BI Application from the App Store. After you have installed the app and signed in with the Power BI account, navigate to the On-Premise TFS Dashboard and see the report created.

⁴ <https://powerbi.microsoft.com/en-us/documentation/powerbi-admin-administering-power-bi-in-your-organization/>

Figure 13 shows the dashboard with the report.

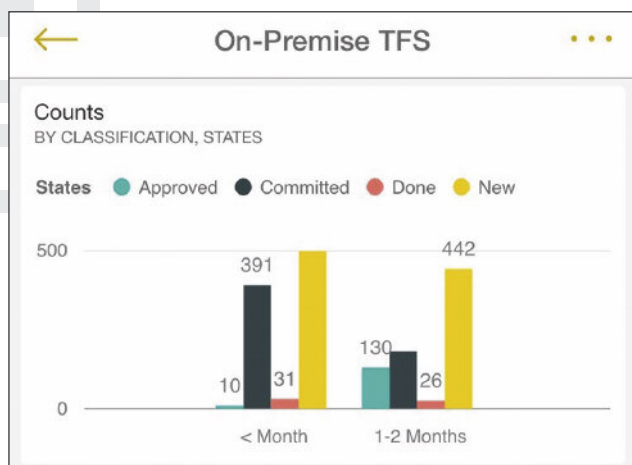


Figure 13: Dashboard in Power BI iPhone App

Clicking on the graph lets you navigate to the details as shown in figure 14.

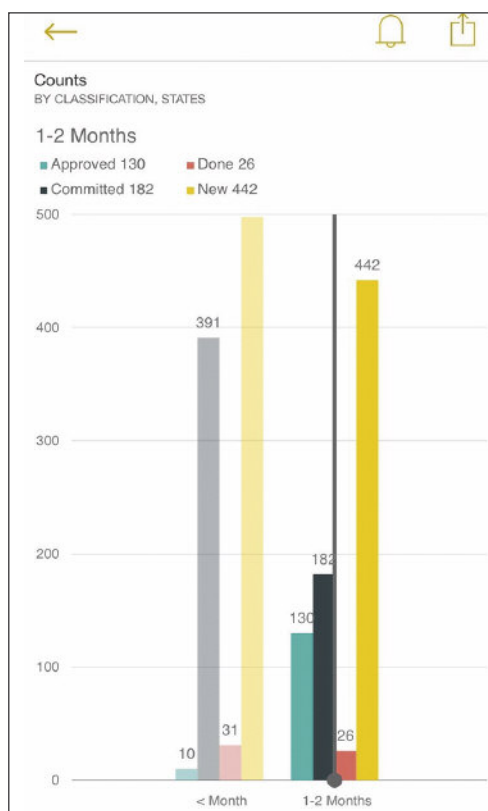


Figure 14: Report details in Power BI iPhone App

Up-to-date information

To meet the stakeholder's requirements, you need to have an

automatic data refresh schedule order to make things easier for the teams. To have the ability of Data Scheduling, you need to have the on-premise data source available for the Power BI environment. A safe way to do so is to use the Power BI Gateway tool. Installation of the gateway is straightforward. Just run the downloadable executable⁵ and configure to connect to your Power BI environment. When the gateway is configured successfully, you will see the configuration as shown in figure 15.

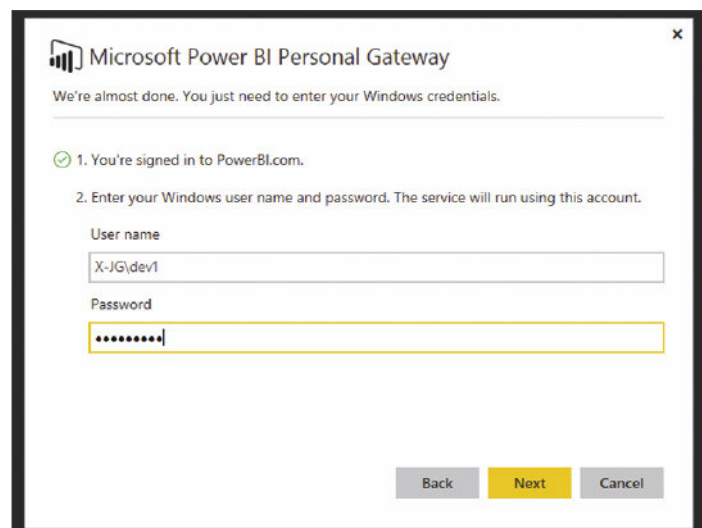


Figure 15: Power BI Gateway configuration

After correctly configuring the gateway you can now use the Power BI Web environment to configure a data refresh schedule. Use the context menu and choose the "Schedule Refresh" option. Figure 16 shows the context menu.

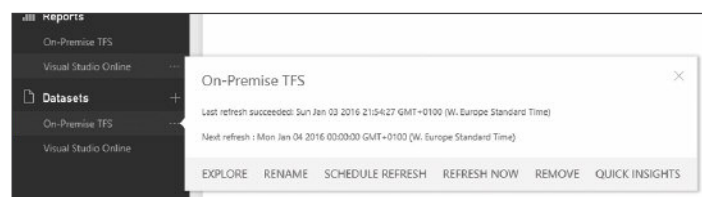


Figure 16: Power BI schedule refresh

This pulls up the settings page. It is possible to schedule a data refresh for a particular dataset. Always check whether the Gateway status is OK, and check whether the Data Source credentials have been provided correctly. If they are correct, configure a desired data schedule refresh. Figure 17 shows the available settings for creating the desired schedule.

⁵ <https://powerbi.microsoft.com/en-us/documentation/powerbi-personal-gateway/>

This enables an automatic refresh of the dataset. You now have completed an end-to-end scenario enabling on-premise TFS warehouse data in an enterprise mobile application! This should definitely satisfy the stakeholder!

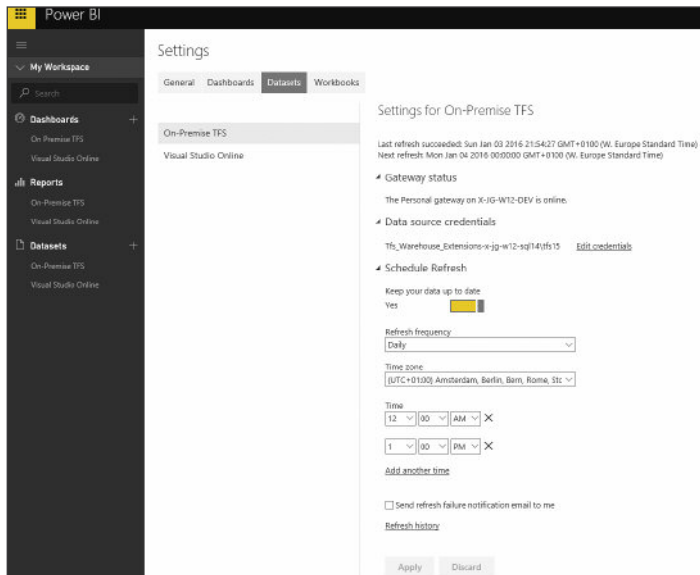


Figure 17: Power BI refresh schedule configuration

Conclusion

Visual Studio Team Services offers powerful extensibility capabilities that allow you to connect to the operational store to create reports. In doing so, you can retrieve data in a different way while the report creation experience is similar. Microsoft recently released a development API for Power BI. Integrating Power BI into your own applications is now possible, and it enables you to develop great reporting solutions. Moreover, Microsoft is working on new functionality in the SQL Server Reporting Services in SQL Server 2016. Enhanced integration between these platforms is to be expected. Content Packs are the solution for distributing dashboards and reports towards other people in your organization. In short, the capabilities of Power BI allow you to really enhance your insights!

Resources and information

- <https://msdn.microsoft.com/en-us/library/dn594433.aspx>
- [https://msdn.microsoft.com/en-us/library/ms181634\(v=vs.80\).aspx](https://msdn.microsoft.com/en-us/library/ms181634(v=vs.80).aspx)
- <https://powerbi.microsoft.com/en-us/documentation/powerbi-personal-gateway/>
- <https://msdn.microsoft.com/en-US/library/ms170438.aspx>
- <http://blogs.technet.com/b/dataplatforminsider/archive/2015/10/29/microsoft-business-intelligence-our-reporting-roadmap.aspx>



JASPER GILHUIS

ALM LEAD CONSULTANT
XPIRIT

Jasper's greatest drive and motivation is process optimization. In addition to processes, he helps users make better use of Microsoft platforms tools such as Team Foundation Server. Jasper provides consultancy, training, and facilitates workshops in the area of ALM and Scrum.

Integrating Protractor UI testing in Visual Studio, TFS and VSTS

Automating software development helps to minimize the development cycle time, and in trying to achieve this, it makes a lot of sense to use the most suitable and effective tools. Out of the box solutions offer several options to test your software. For example MSTest for unit testing, CodedUI for automated UI testing, etc. With the growth of test automation, a huge open source eco system of test tools has become available that have specific advantages when using specific frameworks in applications. This article will show how you can integrate any test tool by adding a little piece of glue code called a test adapter.

Integrating a popular AngularJS UI test tool Protractor

Many applications are built that use AngularJS as the UI framework of choice. When these applications need to be tested, a large number of options is available. You can choose to test the UI using Microsoft CodedUI that is part of Visual Studio, Team Foundation Server (TFS) and Visual Studio Team Services (VSTS). But when you build a user interface with AngularJS, you may prefer to write your tests in the same language as your UI. One of the frameworks that supports this is Protractor. However, Protractor is not with a standard component of Visual Studio and TFS. So let's have a look at how you can change this and make Protractor a fully supported test framework by writing a test adapter. But before you write the adapter, let's take a short look at Protractor.

Introducing Protractor

Protractor is a popular test framework for building UI tests by writing a Spec in a JavaScript file. Before you can run these tests, you need to set up the Protractor toolchain. This toolchain uses Node.js, Jasmine, Selenium and the Protractor tools. To install all the required components, you need to install Node.js (downloadable from <http://nodejs.org>), then you run the following command from the command line using the Node Package Manager (npm) tools:

```
npm install -g protractor
```

This command installs the Protractor tools. The -g flag ensures that the tools are installed for all users on the machine. As a next step, you need to ensure you have the latest Selenium web driver

tools installed on your machine. To do so, you can run the following command from the command line:

```
webdriver-manager update
```

After running these commands you have all the tools you need. When you run Protractor from the command line but a crash occurs, you probably have not installed the Java Virtual Machine on your computer. If this is the case, you also need to install the latest JVM from <http://www.java.com>

Now you are ready to write your first tests. Write Protractor tests in JavaScript and do this by using the Jasmine framework. A Jasmine test is defined by specifying a function that can be used by a **describe** function. Below you see a basic test using the Jasmine framework (describe function) and Protractor elements (browser object):

```
describe('Protractor Demo App', function () {  
    it('should have a title', function () {  
  
        browser.get('http://juliemr.github.io/protractor-demo/');  
  
        expect(browser.getTitle()).toEqual('Super Calculator');  
    });  
});
```

This test is called a spec and it can be run using the command line. You can specify the configuration on the command line. You can also define a configuration file that defines which browser you want to use to run this UI test.

To run this test from the command line, you will need to save the file - to e.g. `firstspec.js` - and then run the following command:

```
protractor --specs firstspec.js --framework jasmine --browser chrome
```

This will run the standalone Selenium server and will use Jasmine as the test framework and the Chrome browser.

The **expect** keyword defines an assertion and will show up in the test results. The result of this test will be: 1 spec, 1 assertion and 0 failures. If you specify an additional option - `resultJsonOutputfile` - you can specify a json file in which the results will be logged.

To learn more about the Protractor framework, follow the tutorials at: <http://angular.github.io/protractor/#/tutorial>

Integrating Protractor into VS, TFS and VSTS

Now that you know about some fundamentals of Protractor, the question remains: how can you integrate this as a first class test framework in Visual Studio, Team Foundation Server and Visual Studio Team Services in the cloud? If you would be able to make this work, you would have best of both worlds.

Fortunately, this is possible by creating a Test Adapter. A Test Adapter fully integrates in the IDE, the test window and the build system of TFS, so you can report the test back as part of your build and release process. Figure 1 shows a conceptual picture of a test adapter.

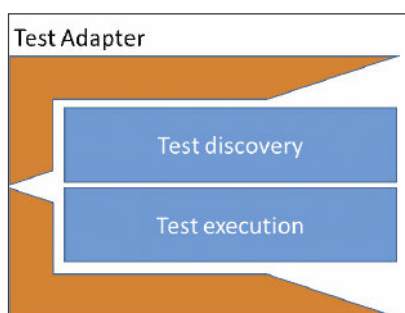


Figure 1: Test adapter

A test adapter provides a common way to discover and run tests. The test discovery part accepts files and checks whether they contain tests and if so, the name of those tests and the test

executer to be used to execute those tests. The test Executer also accepts files and knows how to execute a test it can find in the file. Microsoft introduced these test adapters to provide the flexibility to use any test tool and enable any vendor or open source initiative to fully integrate in their development tools. This is not only limited to Visual Studio, but also allows tests during the build phase on the build server or in the various stages that can be defined in the release pipelines using release management.

Creating a Test Adapter

You can build a Test Adapter by implementing a set of predefined interfaces in a Test Adapter assembly. Start with a simple Class Library Project type, and then add references to the assemblies that define the interfaces and fundamental types to get the integration to work. You need to add references to the following assemblies:

- Microsoft.VisualStudio.TestWindow.Interfaces
- Microsoft.VisualStudio.TestPlatform.Core
- Microsoft.VisualStudio.TestPlatform.Common

You can find these assemblies at the following location:

```
%system drive%\Program Files (x86)\Microsoft  
Visual Studio 14.0\Common7\IDE\CommonExtensions\  
Microsoft\TestWindow
```

Now implement the discovery of tests by creating a class that implements the interface called `ITestDiscoverer`. This interface only has one method to be implemented. It is called: **DiscoverTests**, and gets the following parameters passed when called:

a list of sources, the discovery context, a message logging interface and the `ITestCaseDiscoverySink`.

The list of sources is a list of files that have the required extension. To ensure that only the correct file types are passed, specify an attribute on top of the implementation class. This attribute is called `FileExtensionAttribute`, which is passed in the file extension. In this case you only want JavaScript files as input, so specify `"js"` as extension.

The implementation of the test discovery is shown in code sample 01:

```
[FileExtension(".js")]
[DefaultExecutorUri("executor://ProtractorTestExecutor")]
public class ProtractorTestDiscoverer : ITestDiscoverer
{
    public void DiscoverTests(IEnumerable<string> sources,
        IDiscoveryContext discoveryContext,
        IMessageLogger logger,
        ITestCaseDiscoverySink discoverySink)
    {
        GetTests(sources, discoverySink);
    }
}
```

Code sample 01

One additional attribute you see specified on the implementing class is the **DefaultExecutorUri** attribute. This attribute specifies the unique Uri for the implementation of a class that knows how to execute the tests discovered here. This executor Uri is defined in the class that implements the Executor.

To discover the actual tests, you need to implement the method **GetTests**, which reads the contents of the source file and then checks whether the keyword "Describe" can be found. Describe denotes the start of a test as you have seen in the previous part of this article. If a test is found, you need to create an instance of a type called **TestCase** and pass the name of the test that will be shown in the UI, together with the actual file location and line number where the test was found. If the adapter is used inside the Visual Studio IDE, the **discoverySink** is available and the test case is sent to this implementation. The **discoverySink** will then show the test case in the IDE test window. The basic implementation of this method is shown in code sample 02:

```
internal static IEnumerable<TestCase>
GetTests(IEnumerable<string> sources, ITestCaseDiscoverySink
{
    var tests = new List<TestCase>();
    foreach (string source in sources)
    {
        var testNames =
            GetTestNameFromFile(source);
        foreach (var testName in testNames)
        {

```

```
var testCase = new TestCase(testName.Key,
                            ProtractorTestExecutor.ExecutorUri, source);
        tests.Add(testCase);
        testCase.CodeFilePath = source;
        testCase.LineNumber = testName.Value;
        if (discoverySink != null)
        {
            discoverySink.SendTestCase(testCase);
        }
    }
    return tests;
}
private const string DescribeToken = "describe('";
private static Dictionary<string, int> GetTestNameFromFile(string source)
{
    var testNames = new Dictionary<string, int>();
    if (File.Exists(source))
    {
        int lineNumber = 1;
        using (var stream = File.OpenRead(source))
        {
            using (var textReader = new StreamReader(stream))
            {
                while (!textReader.EndOfStream)
                {
                    var resultLine = textReader.ReadLine();
                    if (resultLine != null && resultLine.Contains(DescribeToken))
                    {
                        var name = GetNameFromDescribeLine(resultLine);
                        testNames.Add(name, lineNumber);
                    }
                    lineNumber++;
                }
            }
            stream.Close();
        }
    }
    return testNames;
}
private static string GetNameFromDescribeLine(string resultLine)
{
    //find describe('
    int startIndex = resultLine.IndexOf(DescribeToken) + DescribeToken.Length;
    int endOfdescription =

```

```

resultLine.IndexOf("'",");
    var testname = resultLine.Substring(startIndex, endOfDescription - startIndex);
    return testname;
}

```

Code sample 02

Now you have the foundation for discovering your tests from JavaScript files. The next step consists of implementing a test. When you select a test in the IDE, a second interface implementation is needed that will execute it. To do so, you need to create an implementation of the interface `ITestExecutor`. This interface consists of `RunTests` and `Cancel`. `RunTests` has two different overloads, i.e. one where a reference is created to the raw source files, the other overload accepts `TestCase` objects as arguments. Both overloads need to do the same thing: just run the test using the information already learned about Protractor tests.

The implementation of the two different method signatures of `RunTests` is rather simple. The implementation is shown in code sample 03:

```

[ExtensionUri(ProtractorTestExecutor.ExecutorUriString)]
public class ProtractorTestExecutor : ITestExecutor
{
    public const string ExecutorUriString = "executor://ProtractorTestExecutor";
    public static readonly Uri ExecutorUri = new Uri(ProtractorTestExecutor.ExecutorUriString);
    private bool Cancelled;
    public void RunTests(IEnumerable<string> sources, IRunContext runContext, IFrameworkHandle frameworkHandle)
    {
        IEnumerable<TestCase> tests = ProtractorTestDiscoverer.GetTests(sources, null);
        RunTests(tests, runContext, frameworkHandle);
    }

    public void RunTests(IEnumerable<TestCase> tests, IRunContext runContext, IFrameworkHandle frameworkHandle)
    {
        m_cancelled = false;
        foreach (TestCase test in tests)
        {
            if (Cancelled)
            {
                break;
            }
        }
    }
}

```

```

}
    frameworkHandle.RecordStart(test);

    var testOutcome = RunExternalTest(test, runContext, frameworkHandle, test);

    frameworkHandle.RecordResult(testOutcome);
}
}
public void Cancel()
{
    Cancelled = true;
}
}

```

Code sample 03

When you receive a call to the `RunTests` overload that accepts a list of sources, you simply call into the previous class that was created to discover the tests in the source files and that will return all the `TestCases` found. After that, call into the second method `RunTests` that accepts the list of `TestCases`.

In the implementation of `RunTests` you need to check whether there was a call to `Cancel` while the tests are executed. To do so, use the flag `Cancelled`. You need to signal the test infrastructure that you started with the run of a single test, and then run the actual tests. When this is finished, signal that this was done by calling `RecordResult` on the `frameworkHandle`. This will show the results in the IDE or will ensure that you get the test output recorded to the *.trx file when you run this from the build infrastructure in TFS.

The final step consists of executing the test. To do so, call the Protractor command line as shown at the beginning of this article. Then specify you want the json result file to record the results. Then parse those results and report this back as the test outcome. The implementation of the test execution is shown in code sample 04:

```

private TestResult RunExternalTest(TestCase test, IRunContext runContext, IFrameworkHandle frameworkHandle, TestCase testCase)
{
    var resultFile = RunProtractor(test, runContext, frameworkHandle);
    var testResult = GetResultsFromJsonResultFile(resultFile, testCase);
    return testResult;
}

```

```

    }

    public static TestResult GetResultsFromJsonRe-
    sultFile(string resultFile,

    TestCase testCase)
    {
        var jsonResult = "";
        if (File.Exists(resultFile))
        {
            using (var stream = File.OpenRead(result-
            File))
            {
                using (var textReader = new
                StreamReader(stream))
                {
                    jsonResult = textReader.ReadToEnd();
                }
            }
        }
        var results =
        JsonConvert.DeserializeObject<List<ProtractorRe-
        sult>>(jsonResult);
        var resultOutcome = new TestResult(testCase);
        resultOutcome.Outcome = TestOutcome.Passed;
        foreach (var result in results)
        {
            foreach (var assert in result.assertions)
            {
                if (!assert.passed)
                {
                    resultOutcome.Outcome =
                    TestOutcome.Failed;
                    resultOutcome.ErrorStackTrace = as-
                    sert.stackTrace;
                    resultOutcome.ErrorMessage = assert.er-
                    rorMsg;
                    break;
                }
            }
        }
        return resultOutcome;
    }

    private string RunProtractor(TestCase test,
    IRunContext runContext,

    IFrameworkHandle
    frameworkHandle)
    {
        var resultFile = Path.GetFileNameWithoutExten-
        sion(test.Source);
        resultFile += ".result.json";
        resultFile = Path.Combine(Path.GetTempPath(),
        resultFile);

        ProcessStartInfo info = new ProcessStartInfo()
        {

```

```

Arguments = string.Format("--resultJsonOutput-
File \"{0}\" --specs \"{1}\" +
                                " --framework jas-
mine", resultFile, test.Source),
        FileName = "protractor.cmd"
    };

    Process p = new Process();
    p.StartInfo = info;
    p.Start();
    p.WaitForExit();
    return resultFile;
}

```

Code sample 04

Additional implementation required for non dll based tests

Since the Protractor spec files are JavaScript files, these files are not tracked by default inside the Visual Studio IDE. To do so, you need to implement some additional infrastructure using Shell Interop and you need to watch the files for changes. This implementation falls beyond the scope of this article, but what you have seen thus far is the full implementation of the real adapter part. This additional work is only needed for Visual Studio integration and the details can be found in the implementation that you can find at the GitHub repo (<http://bit.ly/ProtractorAdapter>), where the full implementation of the Protractor adapter is published. The adapter is available as a NuGet package and as a VSIX integration package for full support in the Visual Studio IDE.

Deploying the Test Adapter in Visual Studio

The test adapter itself is something you can deploy as a NuGet Package and then upload to Nuget.org. This is also what I have done with the Protractor Adapter. I published it as a package on Nuget with the name ProtractorTestAdapter. If you want to use the test adapter in any of your projects, you have to get a reference to the implementation from NuGet. To test the adapter that was just created, you can create a simple web application and from there install the NuGet package with the following package command:

```
Install-Package ProtractorTestAdapter
```

This will add two references to the project. One that is the Protractor.TestAdapter assembly and the second one is the Json.net assembly - this is required because the adapter depends on it for result parsing.

When you build the adapter, you will also find a VSIX package. To get full support in your IDE for the Protractor spec files, you will also need to install this VSIX package. Now you can create a JavaScript file and the adapter will automatically discover the specs from the files. The following screenshot shows a simple web application project with one spec file that specifies the test of the Angular home page:



Figure 2

If you click the Run All option in the test window, you will see the Protractor test run, while the results are reported back to the test explorer window:



Figure 3

So now you have Protractor spec files as first class citizens in your Visual Studio IDE. Because you created a test adapter and provided it as a NuGet Package, it now also automatically integrates into the build infrastructure.

Using the Test Adapter in TFS and VSTS builds

If you commit the ProtractorValidation project to your version control repository, you can define a Build on the TFS server. To show how this is done, I am using Visual Studio Team Services, but this also works for TFS on premises. To define a minimum build that can run your tests, you need to get the sources, then build the solution and finally run the tests.

The build definition is as follows:

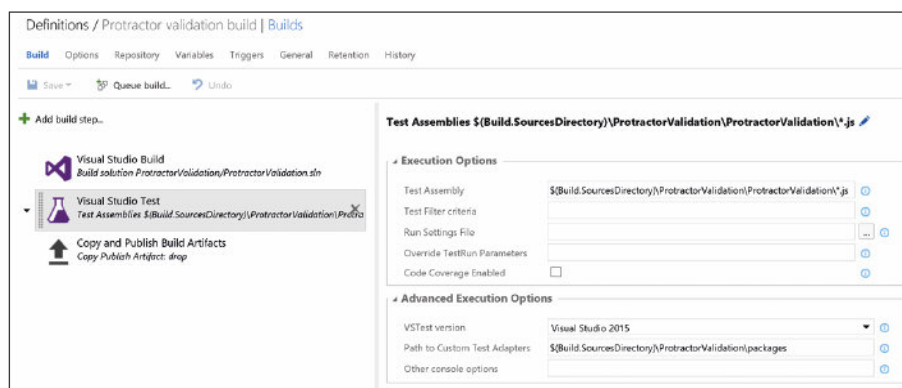


Figure 4

You can see that for the Visual Studio Test step, you had to specify that you are interested in the test files that are JavaScript files. This is why you replaced the default search for dll's by a search for JavaScript files. In the advanced options you need to specify that you want to get the test adapter you referenced using NuGet. Therefore specify a path to Custom Test Adapters to point to the packages folder. This will do a recursive search in all packages downloaded from NuGet.

This includes the custom adapter that will be used. In order to successfully execute the tests, you need a build agent that can run interactively. This can be done by configuring a custom build host. This build host can be a simple Azure machine on which you download the build agent and run the configure command. Here you can specify that you do not want the build agent to run as a service and this enables the agent to run interactively.

Conclusion

This article demonstrated how you can turn a very popular framework for UI testing of Angular websites into an integral part of the daily tools you use. By using the standard extensibility options of Visual Studio, Team Foundation Server and Visual Studio Team Services, you can build a so-called custom test adapter that integrates in the Visual Studio IDE and in the standard build and test infrastructure. Custom Test Adapters allow you to turn any framework you would like to use in your daily build and test cycle into a fully integrated experience. This will make it easy to use these tools and ensures that you don't have to step outside of your daily flow in the IDE. As you can see, Visual Studio, the TFS and VSTS ALM tools have come a long way. Instead of dictating what you need, it is now an open and flexible work environment that can integrate any test tool you would like to use.

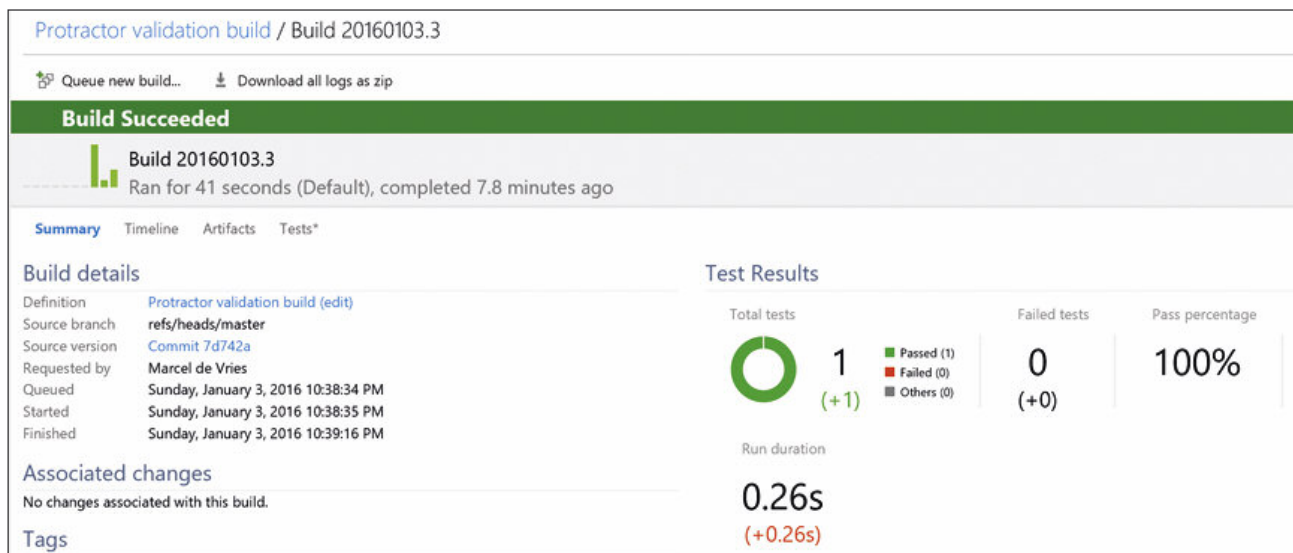


Figure 5

On the build server you also need to install the Protractor tools as described in the beginning of this article. Once you have the interactive build agent registered on your TFS server, you can run the tests as part of your build. When you run this build on your custom build server, you will see the following results as shown in the figure above.



MARCEL DE VRIES

CTO XPIRIT

Marcel spends most of his time finding new technologies and methodologies that help organizations to build superior quality software in a more productive way. Marcel has a passion for sharing knowledge. He is a regular speaker at industry events such as Visual Studio Live, Tech Days, Dev Intersection, etc.



Microsoft
Most Valuable Professional



Microsoft
Regional Director

High availability and disaster recovery in Azure

Continuity is of vital importance for all companies, and today it is extremely important that software runs properly at all times, i.e. 24/7/365. Companies such as Microsoft and Amazon sell their products in many countries all over the world, and at any given time customers are shopping. There's never a good moment for downtime, as downtime immediately costs money. Many companies choose to move their software into the Cloud, but how does this help? Applications must be tolerant to failures at many levels. In order to build resilient, fault-tolerant applications you need to think about two key concepts: high availability (HA) and disaster recovery (DR). In this article you'll learn about these concepts and what the Microsoft Azure Cloud-platform will offer you in this context.

High Availability



Figure 1: System Availability with two parts

The availability of a system is determined by the availability of its parts, and the availability of the entire system is calculated by multiplying the availabilities of each part in a chain of dependencies. Take for example a system with two parts A and B, as shown above. Because part A depends on part B, the availability of this system can be calculated by multiplying the availability of the individual parts:

$$A_{\tau} = A_1 \cdot A_2$$

This leads to a system availability of 98,9%, which means that the overall availability is lower than that of the lowest part! It is, therefore, important to know the availability of each part when you are creating a system that is intended to be highly available.

High Availability is all about eliminating single points of failure in every part of a system by introducing redundancy. This redundancy can be achieved by replicating data and running multiple servers. This increases fault tolerance in order to eliminate outage. In order to know where these single points of failure are located, you need to know about fault domains.

Fault Domains

Consider an application running in a datacenter like Azure. Things may go wrong in different areas: a virtual machine may fail or the entire datacenter could go offline due to a power outage. Fault domains are therefore layered, as shown in figure 2. Obviously this is a simplified view, because every layer comprises multiple other layers, e.g. power supplies, network switches, hypervisors, etc. From this perspective, an application can be made highly available by running it in multiple datacenters – perhaps even using multiple cloud vendors - using multiple virtual machines running in separate server racks.

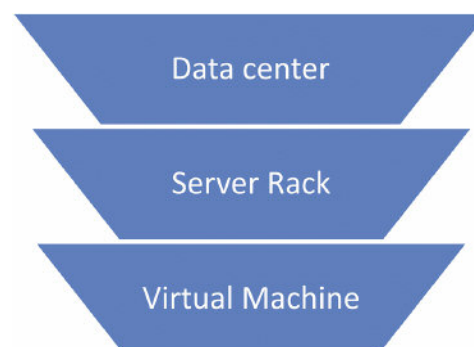


Figure 2: Layered Fault Domains

Disaster Recovery

Disaster Recovery assumes outage and describes ways to deal with an outage. An example of this would be to restore a database backup to a new server after the original database experienced hardware failure.

RTO

A relevant aspect of Disaster Recovery is the maximum accepted time it takes to recover from a disaster until the system is once again fully available. This is called the Recovery Time Objective (RTO). For example, if it takes a day to set up, configure and start using a new database server after a disaster has made the original database unavailable, the Recovery Time would be a day. If the RTO is equal or larger than one day, you meet this objective.

RPO

The maximum accepted amount of data loss after a disaster is called the Recovery Point Objective (RPO). For instance, if a database is (incrementally) backed up every 30 minutes, the Recovery Point value would also be 30 minutes. If your RPO is equal or larger than 30 minutes, you also meet this objective.

Both values are shown in figure 3. Your system should be built and configured to meet the Recovery Point and -Time objectives.



Figure 3: RPO and RTO

If you want mission-critical applications to be highly available and tolerant to failures, both of these values must be close to zero. However, costs will multiply as you move closer to zero.

Operations

Managing High Availability (HA) and Disaster Recovery (DR) for enterprise systems can be a daunting task for IT-pro's. It involves managing large amounts of servers and data, often distributed over multiple datacenters and dealing with large volumes of data. Fortunately the Microsoft Azure platform reduces a lot of this complexity because it offers many HA and DR features as a service. We will now describe some strategies you can use to deal with such disasters and that will help make your mission-critical application highly available and tolerant to disasters. We will do so by using a fictitious web shop as an example project.

ACME Global Shopping

Introducing, 'ACME Global Shopping'. A globally available web shop that 'sells everything'. For ACME it is of vital importance that their web shop is available for shoppers around the globe. They chose Microsoft Azure as a platform for running their software.

Figure 4 shows a simplified schematic overview of the deployment model of their.

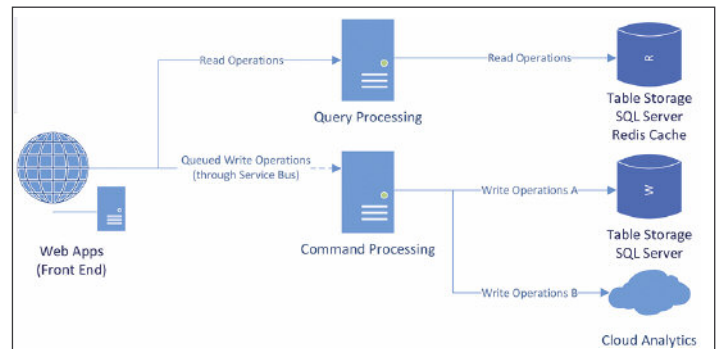


Figure 4: System Conceptual Deployment Model

ACME has a website, which – for write operations – is loosely coupled with two back office systems by means of a queue. Data is then written to several types of data stores and an external analytics service. Read operations are separated from write operations, and are performed on a specialized data store. In order to make its web shop 'highly available', ACME created the

following configurations for the various components in their architecture. When running this architecture on Azure provides you will get some excellent HA and DR features straight out of the box. Some other features, such as the use of the CQRS design pattern (separating reads from writes) were implemented by ACME employees.

Data Storage

ACME faced a number of problems in earlier versions of their systems. All data was simply stored in one SQL Server Database. They were reaching the limits of their server, queries were running slower as the customer base increased and the amount of stored data grew. ACME realized it needed to start storing data based on the characteristics of its use. For instance, the product catalog needs to be highly optimized for search operations, but customer data will mostly be queried by user name. The product catalog changes every day, but data about a customer hardly ever changes. Based on these very different types of use, the most efficient storage type can be selected. For now, we will limit our scope to just two types of storage: relational storage (SQL) and key/value-set storage (NOSQL). When thinking of your own storage policies, you will definitely need to consider document databases as well as blob storage.

SQL

The advantages of using a SQL data store is that it supports transactions and rich querying. It works well for relational data, but unfortunately it has a number of disadvantages: limited performance, limited scalability and relatively high costs. ACME has chosen Azure SQL Database for relational storage, which is Microsoft's Database As A Service Platform. For High Availability, Azure SQL Database data is replicated over three nodes. An SQL transaction only completes when the data is persisted in at least two physically separated nodes. As a result, if a node fails – perhaps due to hardware failure – it can be recovered using the remaining synchronous node without downtime and without resulting in a single point of failure.

To ensure Disaster Recovery, you can create and restore backups of the data, in order to deal with data loss due to application errors or user errors. Creating backups can be done in multiple ways, depending on the selected database pricing tier and of course configuration. Backups are created automatically, and the files are stored in Globally Redundant Storage by default. It is also possible to manually create backups to be stored in a Blob container in a Storage Account.

Restoring backups can also be done in several ways. First, there is Point in Time restore according to which your database is backed up on a regular interval. The backup files are persisted up to 35 days. If required you can restore your database using the backup of any of those intervals. This is often used to compensate for human errors.

Another option is Geo Restore, which will copy database backups to geographically redundant storage. By having your backups available in multiple regions, the system can restore your database in a different region when the primary region becomes unavailable. Finally there is Geo Replication, which comes in two flavors: Standard and Active. Standard Geo Replication will create a replica of your database in the paired Azure region, but this database is not accessible. If the primary region fails, the replica can be used as a fail-over, but this process is not automated. Active Geo Replication is similar to Standard, but allows up to four readable replicas. The read-only replicas provide additional capacity for read-only queries.

NOSQL

The main advantage of using a NOSQL data store is the processing speed. Queries and write operations are generally a lot faster than

their SQL-based counterparts. Also the costs for both transactions and storage are significantly lower. Data is partitioned by default, which results in substantial scalability. The downside is that query support is rather limited. Queries perform best when done using the partition/row keys, in order to avoid table scans across partitions which may reside on multiple machines. In addition, transaction support is limited to records within one partition.

ACME relies on Azure Table Storage for persistence of their non-relational data.

Data stored as part of Storage Accounts is also replicated by default to make it highly available. In this case, one of the options is Locally Redundant Storage (LRS), in which three synchronous copies are made in the same datacenter. You can also configure data to be replicated in an additional region to make it durable even when a complete datacenter goes down. This is called Geo Redundant Storage (GRS). The additional non-readable copies are created asynchronously with an RPO of 15 minutes. This means that there is no additional latency, but if a datacenter goes down, the data in transit could be lost. In this context, it is important to note that different partitions may sync at different speeds, so cross-partition operations on the primary or secondary datacenter may yield different results after disaster. Optionally, the replica data in the paired region can be made accessible for read access (Read Access GRS, or RA-GRS), providing a means to run your application in a reduced functionality mode. However, write operations are not possible at that time.

Only if Microsoft declares a datacenter as being lost will your paired region become the primary region, but there is no SLA on the time between disaster and this decision, so the RTO for this situation is undefined.

If you feel you need to have a close to zero RPO & RTO with full control over your Disaster Recovery while using Table Storage, you will need to write custom code to have clients access multiple datacenters for each write operation and deal with disasters. For inspiration, have a look at Microsoft Research's 'Replicated Table Library for Microsoft Azure Storage' on GitHub . Note that this will increase both latency for write operations and storage costs.

In order to deal with user errors, you could use tools such as AzCopy , which is a command-line utility that will let you copy data to and from Table Storage tables (or blob and file storage). It supports exporting table entities into CSV files, as well as copying data into different regions.

Caching

By carefully thinking about what data store to use, ACME systems is back on track. However, the new architecture still contains SQL Server database servers, which, as we now know, have limited scalability and performance. So, whenever an expensive query is run and the results are reusable for other customers, it makes sense to hold on to the results for a while. The same thing goes for data that is more or less the same for every customer, such as items from the product catalog. Replacing unnecessary queries on the database with queries from memory (caching) saves database server resources and therefore money. Caching can also help make your systems' availability higher because it reduces the dependency on the data stores. The caching strategy itself does not play an active role in Disaster Recovery, but there are caching systems that offer some support to rebuild the cache after disaster by persisting the data durably. In order to keep your system highly available, a cache should not be a single point of failure. This is why distributed caching exists.

For read operations performed by the Query Processing components, ACME put a Redis Cache (Azure's Distributed Cache as a service) in place as one of the Query Stores. The Azure Redis Cache Service is an in-memory cache that can hold up to 530GB of memory. It can be used in three pricing tiers: Basic, Standard and Premium.

The Basic tier has one node - there is no formal SLA for this tier and it runs on shared infrastructure, which is why usage is recommended for non-critical workloads.

The Standard tier consists of two nodes - one master and a slave to make it highly available. There is a 99.9% SLA for this tier.

The most expensive tier is Premium. It builds on the Standard tier and adds the data persistence capability, which periodically stores a snapshot of the entire cache in a (Premium) Storage Account in the same Region, in order to recover from when disaster strikes all nodes. Please note that there is currently no way to rebuild a cache using data from a secondary datacenter, so if the datacenter is lost, the cache will still need to be rebuilt in another way.

The Premium tier also adds Redis Clustering, which enables one cache to be shared across up to 10 nodes, thus allowing more data to be stored. Finally, it adds Virtual Network support to isolate the cache nodes and restrict access to virtual network clients.

From the Standard tier up, when accessing the cache, you are actually talking to a load balancer, which will redirect your operation to the master node. Whenever the master node fails, the load balancer will make sure a healthy slave node becomes master. This makes the cache highly available.

ACME decided to use the Standard tier; having a highly available, distributed disposable cache works sufficiently for their needs.

Service Bus

ACME uses Azure Service Bus for reliable messaging. This will ensure guaranteed ('at least once') delivery of messages from the website to the Command Processing components. This creates a loose coupling, allowing the front end to stay operational even when the Command Processing is not.

When a message is put on the Service Bus and received by the service, the message is first persisted in a messaging store. By default, this is one SQL Database per queue. Just like a custom Azure SQL Database, the data is replicated three times for High Availability. For high performance scenarios a different dedicated messaging store is available, which is similar to the store that is used for Event Hubs.

To make a Service Bus Message Queue more fault tolerant, it can be configured to be a partitioned queue. This will use a separate store for each partition, and messages can be directed into those partitions. Not only will this perform better, but partitions will also serve as backups for each other. When the system fails to store a message in one partition, it will try another. Only if all partitions fail to accept the message, the message is rejected.

Because Service Bus Namespaces are tied to regions, you will need to configure two identical queues in two regions to recover from datacenter-level disasters. Your message-sending application needs to decide whether to put messages in both (active / active) or the last known working namespace (active / passive). For Service Bus Relay only the latter option should be used. Your receiving application will need to be capable of receiving messages from both and must have some de-duplication capabilities (use message id, or correlation id for this). The downside of this approach is having multiple Service Bus resources, which incurs double costs for ownership and usage.

As an alternative, the message-sending application could also opt to store messages in a local store (e.g. MSMQ), until the datacenter becomes available again.

Web Apps

The ACME website is running as a Web App, using multiple instances. Websites run as part of Azure App Services. Several pricing tiers are available here: Free, Shared, Basic, Standard and Premium. Free and Shared allow you to host apps in a shared environment. These very limited environments are intended to be

used for dev/test scenarios and offer no SLA.

Starting from the Basic tier up, there's a 99.95% SLA. It is also possible to configure multiple instances of one app here, which means that single points of failure can be eliminated. Another nice feature is automated backups, which allows backing up files and databases to a storage account. These backups can be restored to recover from a disaster. The Standard tier adds Geo-distributed deployment to make your Web App highly available. This allows the deployment to take place in multiple fault domains at the highest level (datacenter). When used together with the globally available Traffic Manager service (that routes requests to the different datacenters), a fully duplicated environment can be created. Traffic Manager itself is a distributed, highly available service, so it won't be a single point of failure.

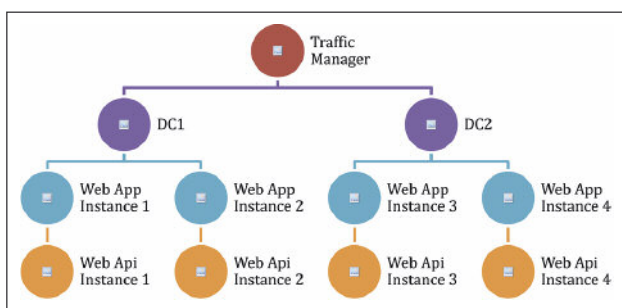


Figure 5: Conceptual architecture (Front End)

You also get up to five deployment slots that can be used in a deployment pipeline. ACME uses this feature to perform automated User Interface testing on a staging environment before taking new versions of their web shop into production, thus preventing data loss due to bugs. The Premium tier adds an isolation feature

called App Service Environment. This provides an isolated, highly available environment for securely running App Service apps on a large scale. They can be created inside a virtual network spanning multiple Azure Regions and can contain up to 50 dedicated computing resources.

Disaster Recovery assumes outage and describes ways to deal with that

Access to and from the virtual network can be controlled on a fine-grained level, significantly increasing application security. For optimal availability performance and security, ACME has opted for the premium tier. Figure 5 shows their conceptual architecture. For the sake of simplicity, only four Web App instances are displayed. In reality, this number would probably be higher for a popular Web site. By putting every App instance into

an App Service Environment, access to the API instances is limited to the Web App calling it. External parties are blocked from accessing it. This adds an additional security layer and prevents unauthorized use of the API.

Cloud analytics

No highly available system is complete without proper analytics. After all, how would you know whether your system is down if nobody is watching it? You don't want to wait for an unhappy shopper to call and tell you your site is not working. You need to respond quickly and effectively at the first signs of trouble, and fortunately Azure also provides a nice feature for this. It is called Application Insights – an analytics service that can monitor your application. It provides two main features: analysis of usage patterns and detection of performance issues. The collected data can be analyzed using the Azure Portal, but also using Microsoft Power BI, which is a great tool to analyze and visualize data. Its SDK allows you to trace usage and error events from every layer in your application and correlate them. Check out the previous issue of this magazine if you want to learn more!

Conclusion

In this article we've discussed a number of ways to make resilient applications that are tolerant to failures on multiple levels, and how using the Azure platform can help you to realize this.



LOEK DUYS

CLOUD CONSULTANT XPIRIT

Loek is an avid Cloud Software Architect who focuses on creating secure, scalable and maintainable systems. He is always looking for ways to leverage the latest additions in the Microsoft stack in order to create even better solutions. He likes helping organizations raise both their technological knowledge and security awareness.

Installing Cloudera on Azure

In order to help a client to explore and analyze large amounts of unstructured data, GoDataDriven installed a Cloudera cluster on Microsoft Azure. In this article we discuss how to install Cloudera on Microsoft's cloud solution. Processing large amounts of unstructured data requires serious computing power, powerful database technology like Cloudera, and scalable infrastructure. A cloud solution, like Microsoft Azure, that is able to scale up and down over time and due to seasonal influences, can be a flexible and cost-effective hosting solution.

Cloudera

Cloudera, founded in 2008, was one of the first organizations to develop an Hadoop distribution. Today, Cloudera delivers a modern data management and analytics platform built on Apache Hadoop and the latest open source technologies. Cloudera Enterprise is the a fast, easy, and secure data platform. Organizations that use Cloudera efficiently capture, store, process and analyze vast amounts of data, empowering them to use advanced analytics to drive business decisions quickly, flexibly and at lower cost than has been possible before.

Microsoft Azure

Microsoft Azure is a cloud service for both infrastructure-as-a-service (IaaS) and platform-as-a-service (PaaS), with data centers spanning the globe.

The following service offerings are relevant when deploying Cloudera Enterprise on Azure:

- **Azure Virtual Network (VNet)**, a logical network overlay that can include services and VMs and can be connected to your on-premise network through a VPN.
- **Azure Virtual Machines** enable end users to rent virtual machines of different configurations on demand and pay for the amount of time they use them. Images are used in Azure to provide a new virtual machine with an operating system. Two types of images can be used in Azure: VM image and OS image. A VM image is the newer type of image and includes an operating system and all disks attached to a virtual machine when the image is created. Before VM images were introduced, an image in Azure could have only a generalized operating system and no additional disks. A VM image that contains only a generalized operating system is basically the same as the original type of image, the OS image. From one VM image you can provision multiple VMs. These virtual

machines will run on the Hypervisor. The provisioning can be done or using the Azure portal or with PowerShell or Azure command line interface.

- **Azure Storage** provides the persistence layer for data in Microsoft Azure. Up to 100 unique storage accounts can be created per subscription. Cloudera recommends Premium Storage, which stores data on the latest technology Solid State Drives (SSDs) whereas Standard Storage stores data on Hard Disk Drives (HDDs). A premium storage account currently supports Azure virtual machine disks only. Premium Storage delivers high-performance, low-latency disk support for I/O intensive workloads running on Azure Virtual Machines. You can attach several Premium Storage disks to a virtual machine (VM). With Premium Storage, your applications can have up to 64 TB of storage per VM and achieve 80,000 IOPS (input/output operations per second) per VM and 2000 MB per second disk throughput per VM with extremely low latencies for read operations. Cloudera recommends one storage account per node to be able to leverage higher IOPS.
- **Availability Sets** provide redundancy to your application, ensuring that during either a planned or unplanned maintenance event, at least one virtual machine will be available and meet the 99.95% Azure SLA.
- **Network Security Groups** provide segmentation within a Virtual Network (VNet) as well as full control over traffic that ingresses or egresses a virtual machine in a VNet. It also helps achieve scenarios such as DMZs (demilitarized zones) to allow users to tightly secure backend services such as databases and application servers.

Deployment Modes To Start A Cluster

At the moment Azure has two deployment modes available:

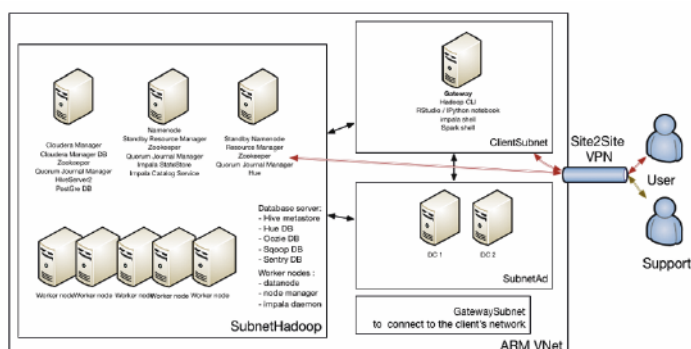
1. ASM (Azure Service Management)

2. ARM (Azure Resource Manager)

The ASM API is the "old" or "classic" API, and correlates to the web portal. Azure Service Management is an XML-driven REST API, which adds some overhead to API calls, compared to JSON. The Azure Resource Manager (ARM) API is a JSON-driven REST API to interact with Azure cloud resources. Microsoft recommends deploying in ARM mode. One of the benefits of using the ARM API is that you can declare cloud resources as part of what's called an "ARM JSON template." An ARM JSON template is a specially-crafted JSON file that contains cloud resource definitions. Once the resources have been declared in the JSON template, the template file is deployed into a container called a Resource Group. An ARM Resource Group is a logical set of correlated cloud resources that roughly share a life span. Using the ARM mode you are able to deploy resources in parallel, which was a limitation in ASM. The new Azure Ibiza Preview Portal is used to provision Azure cloud resources with ARM instead of the ASM API. You are not limited to the portal to deploy your templates. You can use the PowerShell or the Azure command-line interface to manage all Azure "resources and deploy complete templates. The Azure CLI is based on NodeJs and thereby available on all environments. Both ARM and ASM are modes which can be configured using the CLI. Resources deployed in the ASM mode cannot be seen by the resources deployed in the ARM mode by default. If you want to achieve this, you would need to create a VPN tunnel between the two VNets.

Requirements & Design

GoDataDriven's goal was to build a production ready Hadoop cluster, including provisioning machines, that took client specific requirements in account, including enabling single sign-on, the deployment of 3 master nodes to handle load expansions, Cloudera connected to the Active Directory to authenticate users, assure access control using Sentry, install RStudio and IPython on the gateway for analysis. The following installation was designed: The GatewaySubnet is needed to set up the Site2Site VPN between the client's network and the Azure network where the Hadoop cluster resides.



For user management two Active Directory servers were set up in their own subnet, acting also as Domain Name Server.

High traffic between the nodes in the cluster

Because of the high traffic between all nodes in the cluster, the Hadoop machines are in their own subnet. A reason for this is that when you write a file into HDFS, this file is split into blocks (block size is usually 128 MB) and these blocks are placed on the Data-nodes. Each block has a replication factor of 3. Only the master node (Namenode) knows which block belongs to which file. The Namenode does not store blocks, but it does maintain the active replication factor. If a client wants to read a file from HDFS, it will first contact the Namenode, get the location of the blocks and then read the blocks from the Datanodes. The Datanodes send heartbeats to the Namenode and the when the active Namenode notices that a block hasn't got the requested replication factor, it instructs another Datanode to copy that given block.

There is also a ClientSubnet for the machines which can access the cluster. Users can connect to the machines in this subnet, do their analysis, but are not able to SSH to the machines in the Hadoop subnet.

Because of the single sign-on using the Active Directory on the Linux level and configuring Kerberos using Active Directory for the Hadoop services, users can use a single password everywhere.

How To Install Cloudera On Azure?

There are multiple way in which you can install Cloudera on Azure, of two were considered:

1. Install everything from scratch:

- Provision machines and network using Azure CLI
- Use a provisioning tool (like Ansible) to do the Linux configuration
- Install Cloudera Manager
- Install CDH (Cloudera Distribution for Apache Hadoop) using Cloudera Manager

2. Using the ARM template that Cloudera provides to install a Hadoop cluster. This template, available on GitHub, includes OS and network tuning and Hadoop configuration tuning. There is also an Azure VM image, available on the Azure Marketplace, built and maintained by Cloudera which is used during deployment.

Out-of-the-box features of the template:

- Create a Resource Group for all the components
- Create VNet and subnets

- Create availability sets. Place masters and workers in different availability sets
- Create security groups
- o Create Masternode and Workernode instances using the Cloudera VM Image (CentOS image built and maintained by Cloudera). The template automatically uses Azure DS14 machines, which are the only machine types recommended and supported by Cloudera for Hadoop installations.
- For each host a Premium Storage account is created
- Add disks to the machines, format and mount the disks (10 data disks of 1 TB per node)
- Set up forward/reverse lookup between hosts using /etc/hosts file
- Tune Linux OS and network configurations like disable SELinux, disable IPtables, TCP tuning parameters, disable huge pages
- Set up time synchronization to an external server (NTPD)
- Set up Cloudera Manager and the database used by the Cloudera Manager
- Set up Hadoop services using the Cloudera Python API

One of the disadvantages of the template is that it is meant to start up a cluster, but you cannot create extra data nodes and add them to the cluster. The template does not provision a gateway machine for you. After analyzing the gaps between the template provided by Cloudera and the client requirements, a golden middle-way was chosen:

- Use the Cloudera-Azure template to provision the network, set up the machines, configure the OS and install Cloudera Manager
- Use Cloudera Manager (so not the Cloudera-Azure template) to install the CDH cluster.

Best practices for a manual implementation

If you would not use the template, it is advisable to keep the following best-practices in mind:

- When deploying a Linux image on Azure there is a temporary drive added. When using the DS14 machines the attached disk on /mnt/resource is SSD and actually pretty big (something like 60 GB). This temporary storage must not be used to store data that you are not willing to lose. The temporary storage is present on the physical machine that is hosting your VM. Your VM can move to a different host at any point in time due to various reasons (hardware failure etc.). When this happens your VM will be recreated on the new host using the OS disk

from your storage account. Any data saved on the previous temporary drive will not be migrated and you will be assigned a temporary drive on the new host.

- The OS root partition (where also the /var/log directory resides) is fairly small (10GB). This is perfect for an OS disk, but Cloudera also puts the parcels (an alternate form of distribution for Cloudera Hadoop) on /opt/cloudera and the logs into /var/logs. These take up quite a lot of space so a 10 GB disk is not enough. That's why you should move the parcels and the log file to a different disk. Normally the template takes care of this for you. If you install Cloudera without moving these files to a different disk, you will see warning messages in Cloudera Manager that there not enough free disk space available.
- In a distributed system, thus also for a Hadoop cluster (especially if Kerberos is used), time synchronization between hosts is essential. Microsoft Azure provides time synchronization, but the VMs read the (emulated) hardware clock from the underlying Hyper-V platform only upon boot. From that point on the clock is maintained by the service using a timer interrupt. This is not a perfect time source, of course, and therefore you have to use NTP software to keep it accurate.
- If running Linux on Azure install the Linux Integration Services (LIS) - a set of drivers that enable synthetic device support in supported Linux virtual machines under Hyper-V.



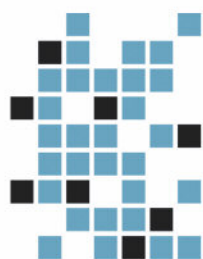
ALEXANDER BIJ
BIG DATA HACKER

Alexander is strong skilled in listening and analyzing issues and working out the best solution. He aims to deliver business value that at least fulfils a clients needs.



TÜNDE ALKEMADE
BIG DATA HACKER

Tünde Alkemade is Big Data Hacker at GoDataDriven, where she focuses on installing, interconnecting and tuning systems and also on implementing software needed for data driven solutions.



GoDataDriven

proudly part of Xebia Group

BIG DATA

Everybody talks about it.

We'd rather let your data speak for itself.

**Big Data and Data Science services
with direct business value.**

Big Data Platform Implementations
Data Science Training
On-site Search Optimization
Real-time Recommendations
Predictive Maintenance
Supply Chain Management

We're proud to provide data services for:

bol.com



Rabobank

ING  BANK



ebay

Schiphol
Amsterdam Airport



wehkamp.nl

Interested in the value of data for your organization?

Visit www.godatadriven.com or call Rob Dielemans: 0031(0)35-6729069

Building a Robot Kit with a Raspberry Pi 2 and Windows 10 IoT Core

The Internet of Things (IoT) ecosystem is growing faster and faster, and with the introduction of Windows 10, Microsoft has made it clear they do not want to be a spectator. Microsoft has already been present since the beginning with the Windows Embedded operating systems, and Windows 10 IoT Core is the next generation OS designed specifically for use in small footprint, low-cost devices and IoT scenarios.

In this article I intend to show how easy it is to use Windows IoT and the Universal Windows Platform.

About Windows 10 IoT Core and the Universal Windows Platform

Windows 10 IoT Core is a version of Windows 10 that is optimized for smaller devices with or without a display. It runs on IoT devices such as the Raspberry Pi 2. Software for Windows 10 IoT Core can be written using the extensible Universal Windows Platform (UWP) API.

In April 2015, a test version of Windows IoT was released, and in November 2015, the official release was announced.

With the introduction of the Universal Windows Platform (UWP), it is now possible to create applications for every device that runs Windows 10. This evolution allows apps that target the UWP to call not only the WinRT APIs that are common to all devices, but also APIs (including Win32 and .NET APIs) that are specific to the device family the app is running on. The UWP provides a guaranteed core API layer across devices. This means you can create a single app package that can be installed onto a wide range of devices.

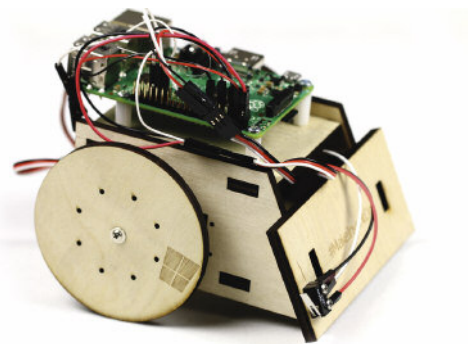
What's more, with that single app package, the Windows Store provides a unified distribution channel to reach all the device types your app can run on.

The Robot Kit

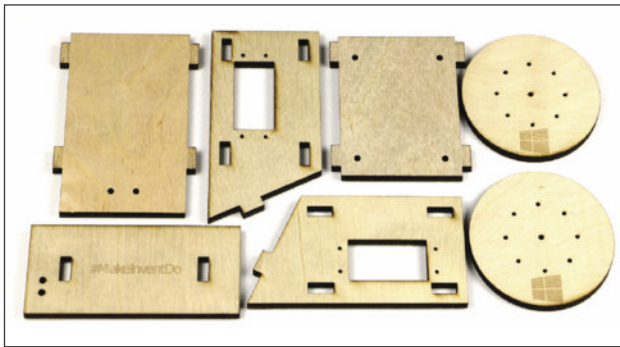
Let's get started

Many people have an IoT device without knowing it. For example, a small Raspberry Pi which they probably use for watching TV. But there is much more that you can do with these kinds of devices.

When you need inspiration for an applicable IoT device, you can look at the Microsoft Windows IoT page on www.hackster.io. A project that really teases everyone's imagination is building a robot, so let's do this! This is how the result of the project should be:



Components needed



Wooden robot frame in 7 pieces



A ball caster (Pololu Ball Caster with 1/2 Metal Ball)



2x continuous rotation servos (SpringRC SM-S4303R Continuous Rotation Servo)



6x 15cm (6") Male-to-Female wires (2 red, 2 white and 2 black)



2x 15 cm (6") Female-to-Female wires (1 red and 1 black)



A set of M2.5 screws, nuts, bolts and standoffs



A digital switch (D2F-01L switch)

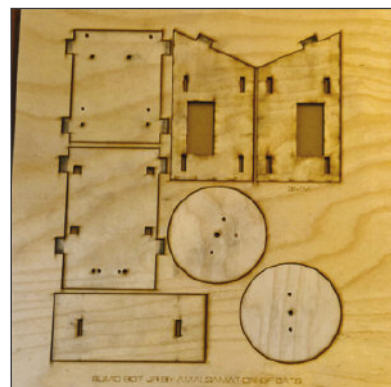


Raspberry Pi 2, a 2 Amp power supply, SD card, network Ethernet cable

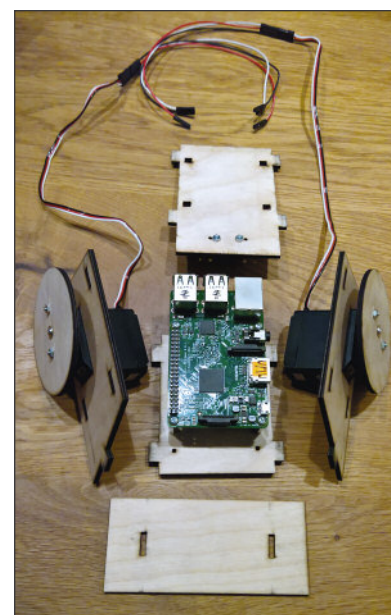
The assembly

For the wooden frame there is a GitHub repo with cutting plans that can be uploaded to an online laser cutting, 3D printing & metal machining services, e.g. Ponoko for the U.S.A or Formular in Europe (Germany). I uploaded the sumbotjr-3mm_ponoko.eps cutting plan from the GitHub repo and used a Plywood Birch 3 mm on a 384x384 mm sheet.

The order process was really easy and it was nice to see the existence of such online services. The final laser-cut package looks like this:



The assembly is really simple - just connect all together like Lego pieces.

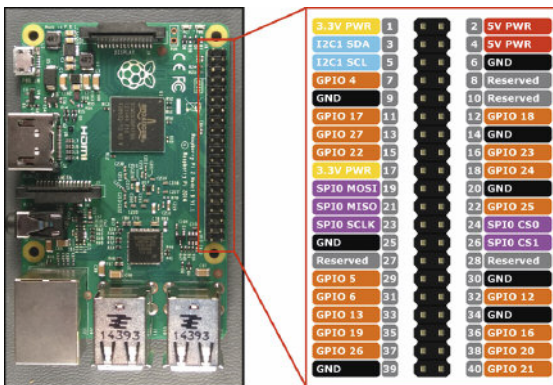


The Pin connections

The following pin layout is available from the Hackster Robot Kit page:

PIN (RPI2)	Name	Function
1	3.3V	3.3V to block sensor
2	5V	5V to Left servo motor
4	5V	5V to Right servo motor
6	GND	GND to Left servo motor
9	GND	GND to Right servo motor
14	GND	GND to block sensor
29	GPIO 5	Output to Left servo motor
31	GPIO 6	Output to Right servo motor
33	GPIO 13	Input from block sensor

I had to figure out which number belongs to a pin, but luckily after some research I found the following images:



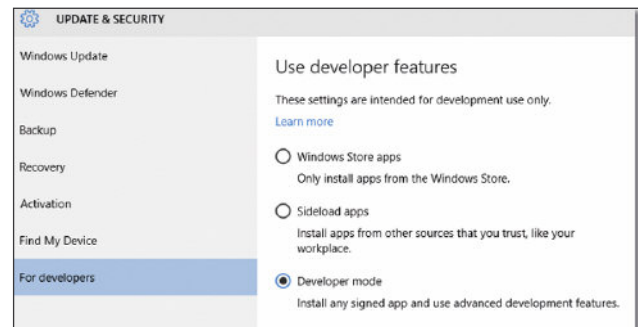
Installing Windows IoT

Set up your development PC

To set up your development PC, do the following:

- Make sure you are running the public release of **Windows 10** (version 10.0.10240) or better.
- Install **Visual Studio 2015 with Update 1** (any version is good, Community, Professional, Enterprise) and be sure to have the Universal Windows App Development Tools installed.

- Install **Windows IoT Core Project Templates**. The templates can be found by searching for Windows IoT Core Project Templates in the Visual Studio Gallery or you can find the link in the references section.
- Enable developer mode on your Windows 10:
 - On your device that you want to enable, go to **Settings**.
 - Choose **Update & security**, then choose **For developers**:



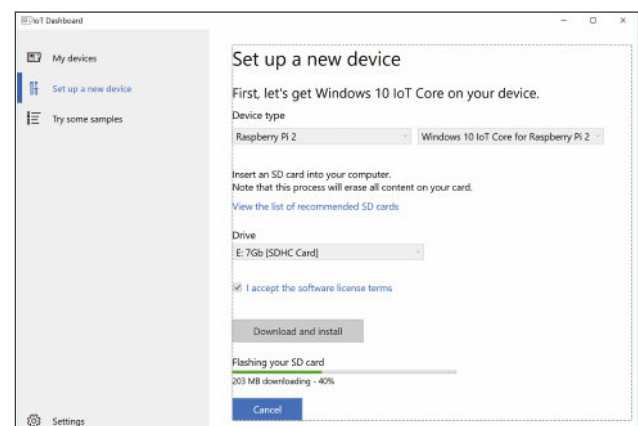
Set up a Windows 10 IoT Core Device

(in our case the Raspberry Pi 2)

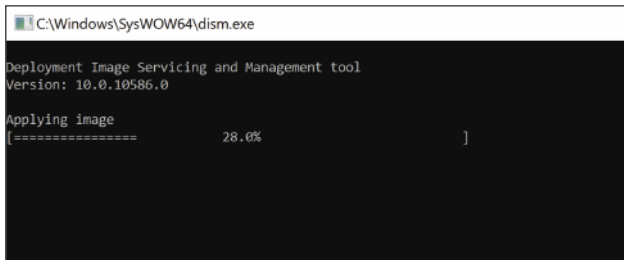
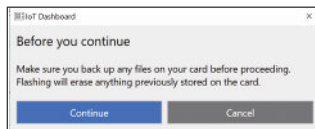
When you have a Raspberry Pi 2, you can set up and configure it easily using the Windows 10 IoT Core Dashboard. The dashboard can be used to set up the RTM (public) version of Windows 10 IoT Core and requires a PC running Windows 10. I added a link to the dashboard application in the References section. Put your MicroSD card into your PC and carry out the following steps:

- Start the **IoT Dashboard** application.
- Click on **Set up a new Device**.
- Select Device Type: **Raspberry Pi 2** and **Windows IoT Core for Raspberry Pi 2**.
- Click on **I accept the software license terms**.
- Click on **Download and Install**.

The dashboard will start to download the Windows IoT image:



After the download has been completed, it will ask to install it on the microSD card:



After the image has been written on the microSD, safely remove it from your PC and insert it into the Raspberry Pi.

Connect your Windows 10 IoT Core device to your development PC:

In order to develop apps for your IoT device, the IoT Core device and development PC should be on the same local network. There are a few options for setting this up.

Option 1: Plug your device into your local network

The easiest and best way to connect to your device is to plug it into a local network that your development PC is already connected to. Plug the Ethernet cable from the device into a hub or switch on your network. To keep things simple, it's best if you have a DHCP server (such as a router) present on your network so the device gets an IP address when it boots.

Option 2: Connecting your Windows 10 IoT Core device directly to your PC & setting up Internet Connection Sharing (ICS)

If you don't have a local network to plug your device into, you can create a direct connection to your PC. In order to connect and share the internet connection in your PC with your IoT Core device, you must have the following:

- A spare Ethernet port on your development machine. This can be either an extra PCI Ethernet card or a USB-to-Ethernet dongle.
- An Ethernet cable to link your development machine to your IoT Core device.

Follow the instructions below to enable Internet Connection Sharing (ICS) on your PC:

- Open up the control panel by right-clicking on the Windows button and selecting **Control Panel**, or by opening up a command prompt window and typing **control.exe**.
- In the search box of the control panel, type **adapter**.
- Under Network and Sharing Center, click View **network connections**.
- Right-click the connection that you want to share, and click **Properties**.
- Click the Sharing tab, and select the **Allow other network users to connect through this computer's Internet connection box**.

After you have enabled ICS on your PC, you can now connect your Windows 10 IoT Core device directly to your PC. You can do this by plugging in one end of the spare Ethernet cable into the extra Ethernet port on your PC, and the other end of the cable into the Ethernet port on your IoT Core device.

Note:

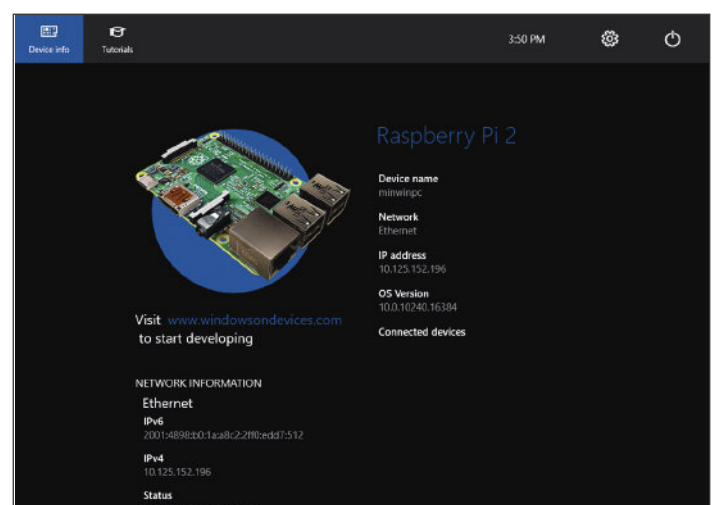
The Sharing tab won't be available if you have only one network connection.

Boot Windows 10 IoT Core

If you have an HDMI cable and a monitor with HDMI input, connect it to your Raspberry Pi. This is not required, but it makes it a lot easier to see what is happening.

Now it's time to connect the power adapter to the Raspberry Pi, after which it will start loading the Windows IoT image on the microSD card. This can take some time, so be patient.

If you have connected the Raspberry Pi with a HDMI cable to a monitor, the following information will appear on the screen:



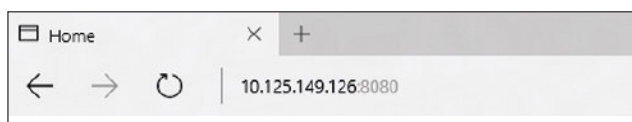
As you can see, the IP address of the Raspberry Pi is shown. Note it because we will need it.

If you have not used the HDMI connection, you will need to open the Windows IoT Dashboard application. To do so, go to My Devices and after waiting a number of seconds, it will detect the Raspberry Pi, and show you the IP address:



Connect to the Windows Device Portal through your browser

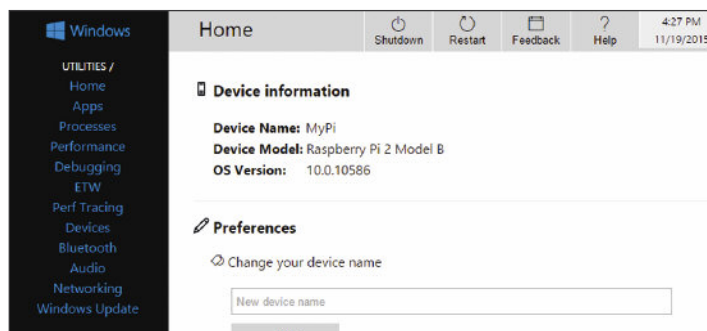
Enter the IP address into the address bar. Add :8080 onto the end.



In the credentials dialog, use the default username and password:
Username: Administrator
Password: p@ssw0rd

The Windows Device Portal should launch and display the web management home screen!

You can also launch the Windows Device Portal tool from the Windows IoT Dashboard application by clicking on your device, and clicking on Open in Device Portal.



Installing the Robot Kit app

The Robot Kit application is on Github, so we need to clone the repo. Open Visual Studio and clone the following git address:

<https://github.com/ms-iot/build2015-robot-kit.git>

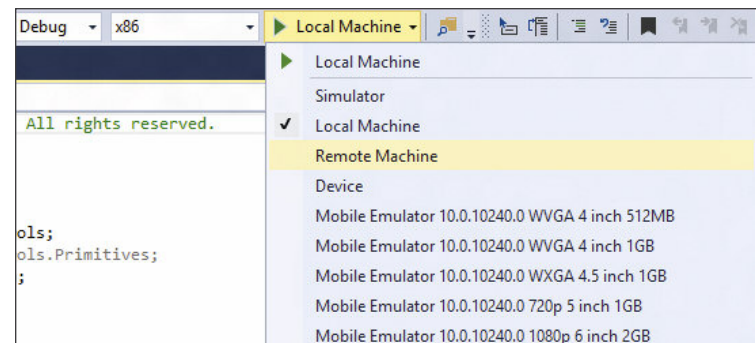
Open the solution, search for the following string:

```
private static String defaultHostName = "tak-hp-laptop";
```

and change the string into your PC's IP address (not the Raspberry Pi address!).

Deploy Your App

- With the application open in Visual Studio, set the architecture in the toolbar dropdown to **ARM**.
- Next, in the Visual Studio toolbar, click on the **Local Machine** dropdown and select **Remote Machine**.

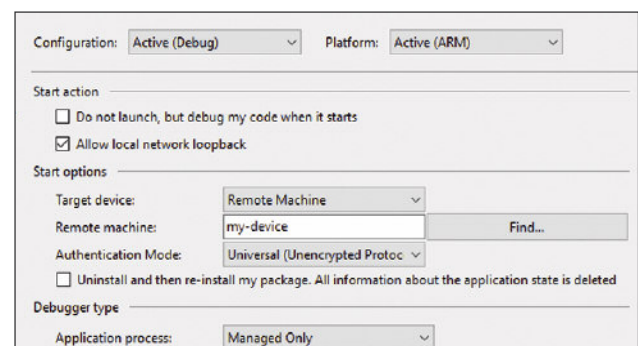


At this point, Visual Studio will present the **Remote Connections** dialog. Use the IP address of your Raspberry Pi.

After entering the device IP, select **Universal (Unencrypted Protocol)** Authentication Mode, then click **Select**.

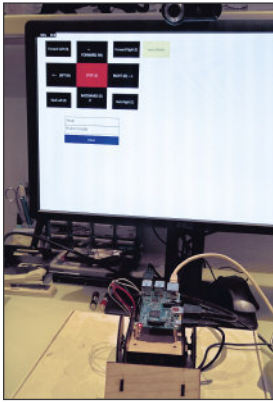


You can verify or modify these values by navigating to the project properties (select Properties in the Solution Explorer) and choosing the Debug tab on the left:



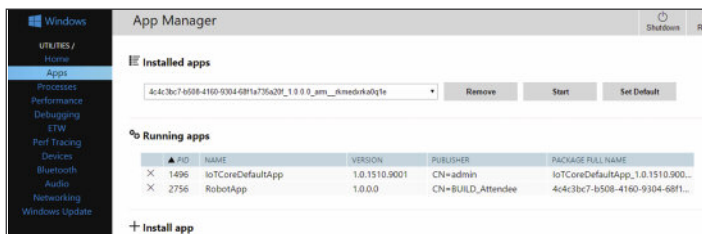
When everything has been set up, you should be able to press F5 from Visual Studio. If there are any missing packages that you did

not install during setup, Visual Studio may prompt you to acquire those now. The Robot Kit app will deploy and start on the Windows IoT device:



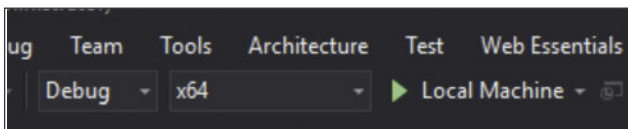
If you attach a USB keyboard to the Raspberry Pi and you press one of the keys shown in the app (W, A, D, X, Z, C, E, Q, S), the robot will start to move!

If you surf on the Windows Device Portal and click on **Apps**, you will see that the Robot App has been deployed and is running:

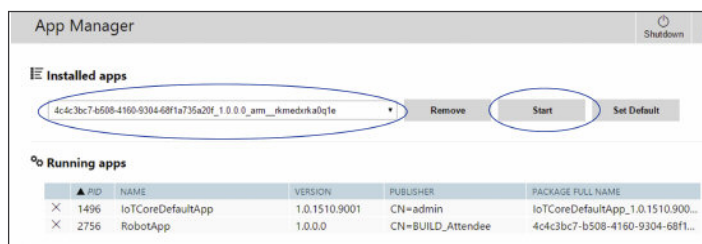


Run the application locally

From Visual Studio, stop the running application pressing the stop button. In the configuration manager dropdown, choose **x86** or **x64**, and **Local Machine**:



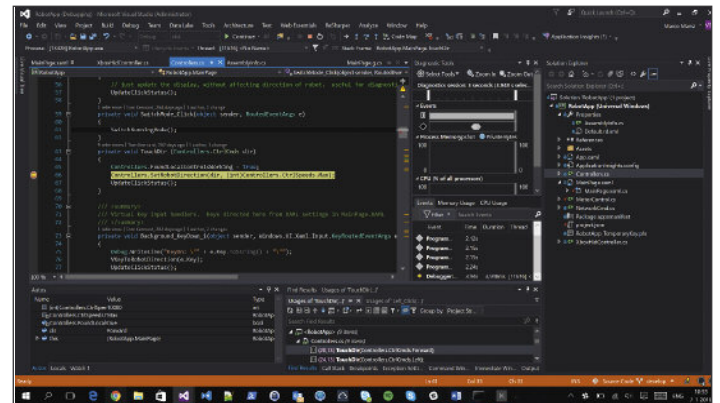
Press **F5** and the application will also run from your PC! This is because it is a Universal Windows Application, which can run on every device running Windows 10 without changing a single line of code. Leave the application running, go back to your browser and go to the Windows Device Portal, click on **Apps** and from the dropdown menu, select the Robot App (it should be something with a strange GUID name in it), and press start:



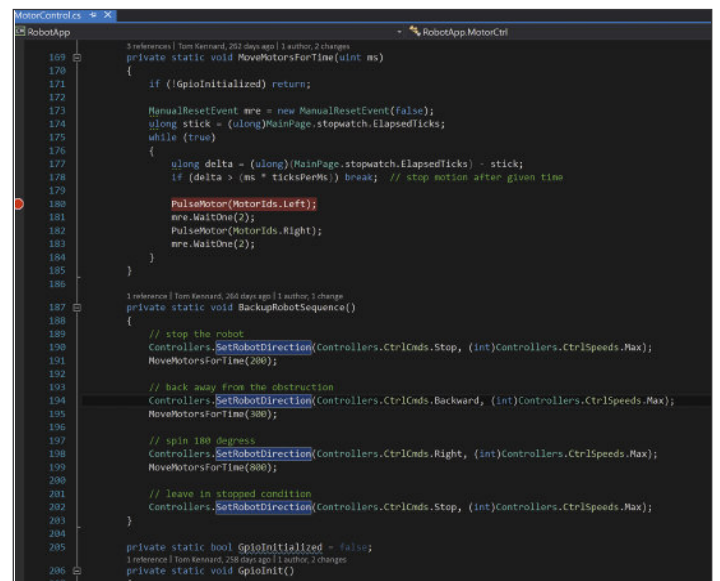
From the Robot App running on your PC, again press one of the buttons (W, A, D, X, Z, C, E, Q, S) and you will see that the Robot will move again. However, now it is controlled from your PC!

Visual Studio

From Visual Studio you are now able to debug what's happening on the IoT device:



As you can see, the code is written in C#, which means everything is managed, and there is no need to write in a lower level language such as C or C++ to talk with the hardware.



Conclusion

This article has demonstrated how Windows 10 IoT Core can be used to work with low cost devices, and how the UWP ecosystem makes it a lot easier to write software that can run in different hardware environments without changing a single line of code.

We didn't dig into the code, but it certainly is worth checking it out so you can learn how it works.

There is a starter pack that you can buy, and which contains all the items required to learn the basics of programming with Windows IoT devices. The relevant link is included in the references section.

The Internet of Things is becoming more and more popular, with business companies creating new devices every day, for example wearables (smart bands, watches etc....) and domotica components (smart thermostats, security systems etc....). With Windows 10 IoT Core and the Universal Windows Platform there is now a unified ecosystem that allows you to easily create software that can interconnect with different devices, making it easier to focus on business functionality.

References:

The following links are the sources for my article, or the sites where I refer to.

- Starter Pack for Windows 10 IoT Core on Raspberry Pi 2
<http://ms-iot.github.io/content/en-US/AdafruitMakerKit.htm>
- Get Started with Windows IoT
<https://ms-iot.github.io/content/en-US/win10/ConnectToDevice.htm>
- Connect your Windows 10 IoT Core device to your development PC
<https://ms-iot.github.io/content/en-US/win10/ConnectToDevice.htm>
- Windows 10 IoT Core Dashboard
<http://go.microsoft.com/fwlink/?LinkID=708576>
- Set up a Raspberry Pi 2
<https://ms-iot.github.io/content/en-US/win10/RPI.htm>
- Microsoft Hackster page
<https://microsoft.hackster.io>
- Robot Kit Hackster page
<http://www.hackster.io/windowsiot/robot-kit>

- Robot Kit Source Code on Github
<https://github.com/ms-iot/build2015-robot-kit/>
- Servo Pin-out
<http://imgur.com/a/tDZY5>
- Cutting plans
https://github.com/makenai/sumobot-jr/tree/master/cutting_plans
- Ponoko locations
<http://www.ponoko.com/about/contact>
- Formulor
<http://www.formulor.de/>
- Windows IoT Core Visual Studio Project Templates
<https://visualstudiogallery.msdn.microsoft.com/55b357e1-a533-43ad-82a5-a88ac4b01dec>
- Windows Device Portal
<https://ms-iot.github.io/content/en-US/win10/tools/DevicePortal.htm>



MARCO MANSI

CLOUD CONSULTANT XPIRIT

Marco is dedicated to anything involving technology, with a focus on software development and architecture. He is curious and interested in new developments, quickly investigating their potential and possible implementation in the real world. Marco loves open source and thinks that sharing knowledge is the key to improvements.



YEAR ANNIVERSARY



37 FUN EVENINGS



Windows
Azure

175,200 MINS



36.9 AVG. AGE
OF EMPLOYEES



51



DEVICES IN USE

14



PROJECTS
CONTRIBUTED

17

CONFERENCES



SATISFIED
CUSTOMERS

10101010101011
01011011110001
111101100101110

1,343,959

LINES OF CODE

0101101111000
11110110010111
0101101111000

2.3
COMPUTERS
PER PERSON



6

MVPS
REGIONAL
DIRECTORS ETC



183,498
MILES FLOWN



8395

CUPS OF
NESPRESSO



161 YEARS
OF EXPERIENCE



1

BABIES BORN

CONFERENCE
PRESENTATIONS

46

ATTENDEES
SERVED

5,749



481,800 KMS
TRAVELED BY CAR



Think ahead. Act now.

Xpirit is the youngest 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



Think ahead. Act now.