

Xpirit Magazine

Patriek van Dorp
An Introduction to
Azure Service Fabric

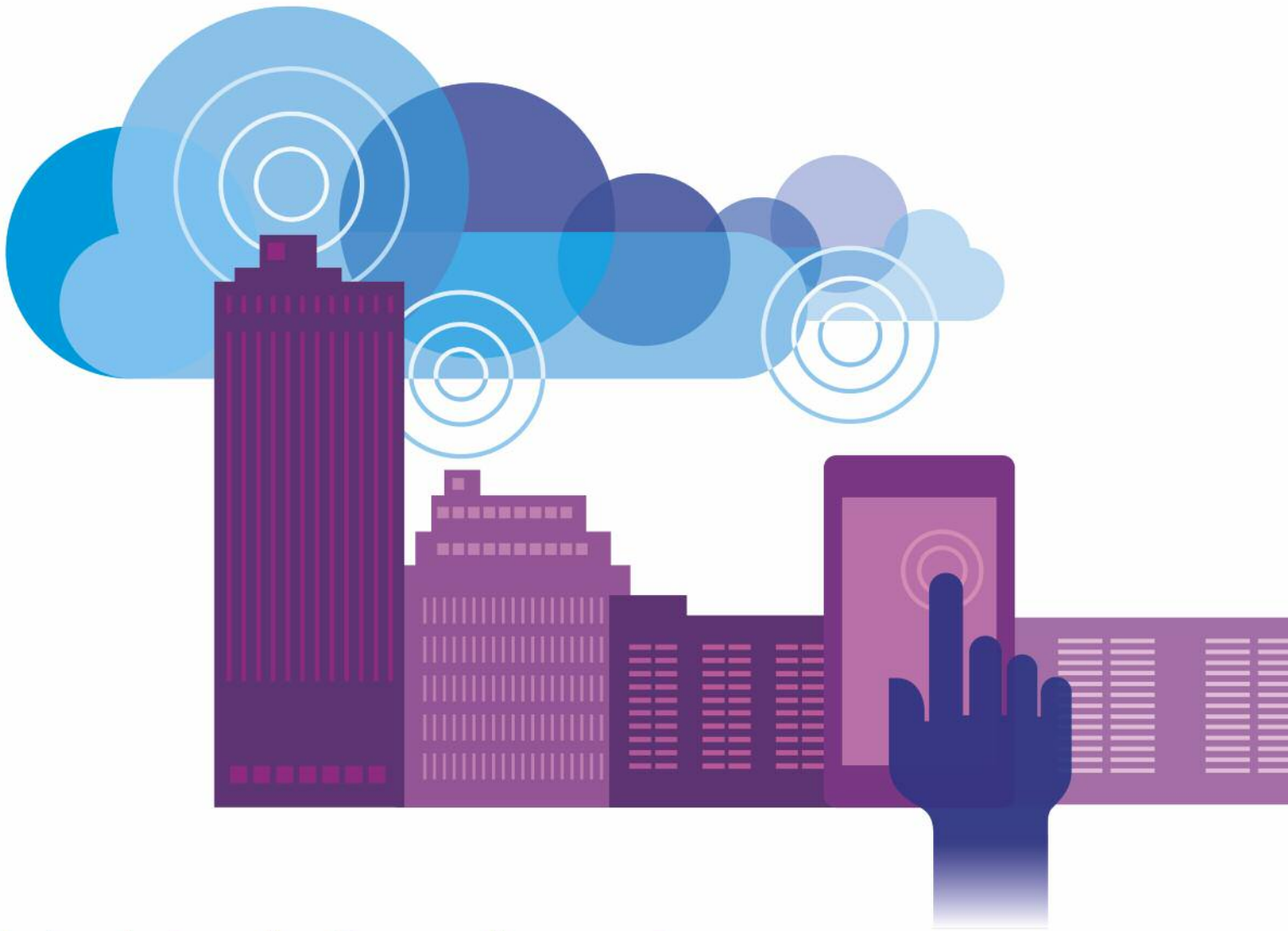
Alex Thissen
First programming
experience with
Microsoft HoloLens

Roy Cornelissen
Lessons learned:
migrating an N-Tier
web app to microservices

Other articles are: ■ Using Microsoft Application Insights to Implement a Build, Measure, Learn Loop - Marcel de Vries
■ Develop Cross-platform mobile hybrid apps with Ionic - Pascal Naber ■ Git vs Team Foundation Version Control: Getting started - Rene van Osnabrugge ■ Start with Visual Studio Release Management vNext - Rene van Osnabrugge
■ API Managment - Marcel Meijer



Think ahead. Act now.



Verbind. Synchroniseer. Transformeer de wereld van apps.

Welkom in het tijdperk van buitengewone verwachtingen. Want elke app moet altijd soepel op ieder apparaat kunnen werken. Zonder haperingen en problemen. Visual Studio 2013 helpt ontwikkelaars te presteren met de meest geavanceerde én geïntegreerde oplossing die er op de markt is: een ultramoderne set met tools en services die u helpen bij het ontwikkelen, testen en implementeren van servicegerichte apps. De gebruikservaring wordt moeiteloos getransformeerd op alle Windows-apparaten.

Ontdek Visual Studio 2013
www.visualstudio.com

Het zijn mooie tijden voor ontwikkelaars

Colofon

Edition:
Xpirit Netherlands bv
No. 1 • May 2015

Editorial Office:
Xpirit Netherlands bv

This magazine was made
with the help of:
Pascal Greuter, Marcel de Vries,
Patriek van Dorp,
Rene van Osnabrugge, Pascal Naber,
Roy Cornelissen, Alex Thissen and
Marcel Meijer.

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

©2015 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 permis-
sion of the Xpirit Netherlands bv.

All trademarks are property of their
respective owners.

Advertisement

Microsoft	2
Xebia University	29
Xpirit Netherlands bv	44

Welcome to Xpirit Magazine!



With great pride I welcome you to the very first edition of Xpirit Magazine. Our journey to make a real difference in IT started in November 2014, when Xebia and a group of very talented people came together to found Xpirit, a new company that always puts people first, and chooses to only work with the very best so our customers achieve more.

At Xpirit we firmly live to our core principles of People First, Quality Without Compromise, Customer Intimacy, and Sharing Knowledge. As the foundation of our organization, our core principles ensure we stay focused on helping our customers reach their full potential, while we get to do the work we love. At Xpirit we are more than just a consultant, we are always looking towards the future. And because we take our profession seriously, we invest more time to really understand our customers, and seek out innovative solutions that will help them succeed. We do make a difference, because that is what we are: different!

This Xpirit Magazine is an expression of one of our core principles: sharing knowledge. You'll find in-depth articles on emerging technologies that only a select few have experienced. That's because at Xpirit our team lives at the cutting edge (like our article on HoloLens), it's what makes our people tick and it's a competitive advantage that will make your organization stand apart.

Along with publishing articles, you'll see us at conferences around the world where we present sessions and workshops in the domains of cloud-based solutions, application lifecycle management and enterprise mobility. You might already have met members of our team at Microsoft Tech-Ed Europe, Visual Studio Live Las Vegas, Visual Studio Live San Francisco, Techorama Belgium, Techdays in The Hague or DevIntersection in Amsterdam. Included in this first edition of Xpirit Magazine are articles on mobile cloud development and Azure stack, along with our first in-depth experience on software development using the brand-new Microsoft HoloLens. There's also a great article on Git versus TFS that critically examines the competitive advantages of both new and proven technologies. You'll get plenty of food for thought with our vision on migrating applications to a microservices architecture and the best way to start with Visual Studio Release Management, as well as new ideas to take your IT and business to the next level with our insights into implementing a build-measure-learn loop. At Xpirit, we really understand your business!

We wish you a pleasant and informative time reading this magazine, and look forward to hearing from you.

Pascal Greuter
Managing Director Xpirit





Table of content

■ Welcome to Xpirit Magazine	3
■ Table of content	4
■ Using Microsoft Application Insights to Implement a Build, Measure, Learn Loop	5
■ An Introduction to Azure Service Fabric	10
■ Develop Cross-platform mobile hybrid apps with Ionic	15
■ Git vs Team Foundation Version Control: Getting started	21
■ First programming experience with Microsoft HoloLens	25
■ Lessons learned: migrating an N-Tier web app to microservices	30
■ Start with Visual Studio Release Management vNext	35
■ API Managment	40

Using Microsoft Application Insights to Implement a Build, Measure, Learn Loop

In our industry we are constantly striving to build better software faster. At the moment you see that continuous delivery is the current technology hype where we all try to deliver the software in faster cycles to the customer and try to achieve to build only the software that brings the highest business value to the customer. Many companies are now selling consulting services and products that focus primarily around the delivery and installation of the software products in test and production environment so the value actually reaches the customer in a timely manner. This is all great and does deliver a lot of value but there are still a lot of issues we need to solve. One of the primary issues we still need to solve is how we can determine what additional features will actually bring the most value to the end user.

It is crucial to know that you are delivering the right product that actually provides value. You want to know the performance of our application in production and how specific usage of the system can impact the performance of our application. If something goes wrong you want to get diagnostic information from your system in production so you can see or even predict when things will break. All these questions are solvable, but most of the time they are solved with custom point solutions that take up a lot of time to build and maintain. In this article I will show you how a product called Microsoft Application insights can provide these insights with out of the box functionality so you can focus on building features instead of worrying about these kinds of infrastructural fundamentals in your application.

Build, Measure, Learn

Application Insights collects, processes and presents a wide variety of telemetry data including performance, usage, availability, exceptions, crashes, environment, log and developer-supplied data from all components of an application—including clients (devices and browser), servers, databases and services. With this "360 degree view" of your application, Application Insights can quickly detect availability and performance problems, alert you, pinpoint their root cause and connect you to rich diagnostic experiences in Visual Studio for diagnosis and repair. It also supports continuous, data-driven improvement of an application. For example, it highlights which features are most and least used, where users

get "stuck" in an application, where and why exceptions are occurring, which client platforms are being used with which OS versions, and where performance optimizations will make the biggest impact on compute costs. By incorporating Application Insights into your product you can start gaining insights in the usage of your product in real time, giving you information that you can use to better decide what needs to be done next in your development team.

Application Insights provides libraries you can include in virtually any type of application. There are support libraries for native Java applications, iOS applications (Objective-C), Android applications, Windows Store applications and web applications. You can very easily gain access to these libraries by either pulling them from GitHub, or for .NET applications by getting them from NuGet.

At the moment Microsoft is transitioning from Application Insights that was part of the Visual Studio Online offering to an offering that is an intrinsic part of the Microsoft Azure offering. For this reason you will find two types of libraries, the old and the new ones. The one I describe here are the new libraries that provide a common API regardless of the platform you target. So methods I show here in a web application are exactly the same on the other supported platforms. At this moment you get the libraries as pre-release packages from NuGet.

Measuring application performance

With Application Insights you can get diagnostics on both the client-side performance and the server-side performance of e.g. your web application. The way to get started and get performance data is first by adding Application Insights into your project.

In my example I will use an ASP.NET MVC website, but this works the same for the other application types I just mentioned. You can use the Visual Studio Wizard to generate a lot of stuff for you. I will describe what to do by hand so you can also apply it to projects you manage without Visual Studio. First, you need to go to the new Azure portal and there create a new Application Insights profile.

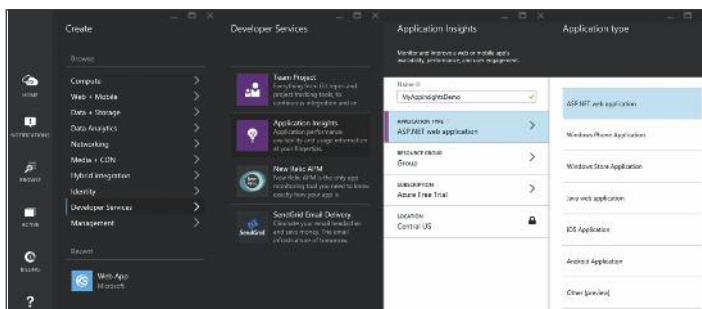


Figure 1 - Create Application Insights Profile

After specifying the correct Application Insights name, application type, resource group and region you will get the dashboard where all your telemetry data will be shown when we have set up things correctly in our application.

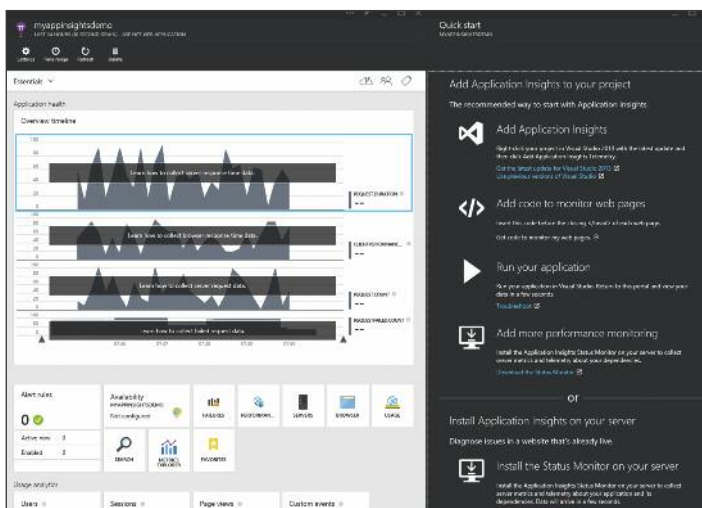


Figure 2 - Initial Application Insights Dashboard

As you can see in the portal, you can now add Application Insights into your project in Visual Studio and add code to your web page to show the client performance of your web application. The server-side code is nothing more than registering Application Insights, and it will start intercepting all work done in your server-side code, including the detection of outgoing dependencies and the time consumed in those calls. When you add Application Insights to your ASP.NET MVC application it will register an Http Module in the web.config that will initialize Application Insights and start the telemetry dataflow of your application. On other platforms like e.g. mobile applications, you need to bootstrap Application Insights in the startup of your application.

Application Insights is configured using a file called ApplicationInsights.config. This is also the location where you specify the InstrumentationKey that identifies where your data needs to flow. You can keep this key the same for all instances of your website, so all data flows to the same location on Azure. After setting up these basics you will already get all the basic server performance and usage data flowing to the portal dashboard. You can see the first charts already light-up in your portal.

With websites that have heavier JavaScript on the client it is important to also keep track of the client side performance these days. You can also enable client-side tracking of performance by injecting a small piece of JavaScript into your application. This is more or less the same concept as you have with other tools that e.g. track user behaviour on pages, like Google Analytics. Each page must have this small piece of JavaScript code that you can copy from the portal. In an MVC app the most convenient place to insert this code is in the master template file that is used by every page. This is standard _Layout.cshtml file that can be found in the "views\Shared" folder. The script you need to place in the page can be copied from the portal when you click on the "Add code to monitor web pages" blade in the Azure portal. The code is shown in the following code snippet:

```
<script type="text/javascript">

    var appInsights=window.appInsights || function(con-
fig){

        function s(config){t[config]=function(){var
i=arguments;t.queue.push(function(){t[config].apply(t,
i)}})}var
t={config:config},r=document,f=window,e="script",o=r.c
reateElement(e,i,u;for(o.src=config.url||"//az416426.
```

```

vo.msecnd.net/scripts/ai.0.js",r.getElementsByTagName-
Name(e)[0].parentNode.appendChild(o),t.cookie=r.coo-
kie,t.queue=[],i=["Event","Exception","Metric","PageVi-
ew","Trace"];i.length;s("track"+i.pop());return con-
fig.disableExceptionTracking||(i="onerror",s("_"+i,u=
f[i],f[i]=function(config,r,f,e,o){var
s=u&&u(config,r,f,e,o);return s!==!0&&t{"_"+i}(con-
fig,r,f,e,o),s)),t
}({
instrumentationKey:"841abd52-569e-4c2f-a213-
32ded088cdb2"
}));

window.appInsights=appInsights;
appInsights.trackPageView();
</script>

```

In this piece of JavaScript code I highlighted two parts. First where you supply the instrumentation key. This is the same key as you will find in the ApplicationInsights.config file. The second part is the initialization of Application Insights in the client. You see you can just reference a variable named appInsights and on that you can call a variety of methods to track information from the client. The trackPageView() call will now be executed on each page the client loads and that will generate the basic performance data you can see in the portal.

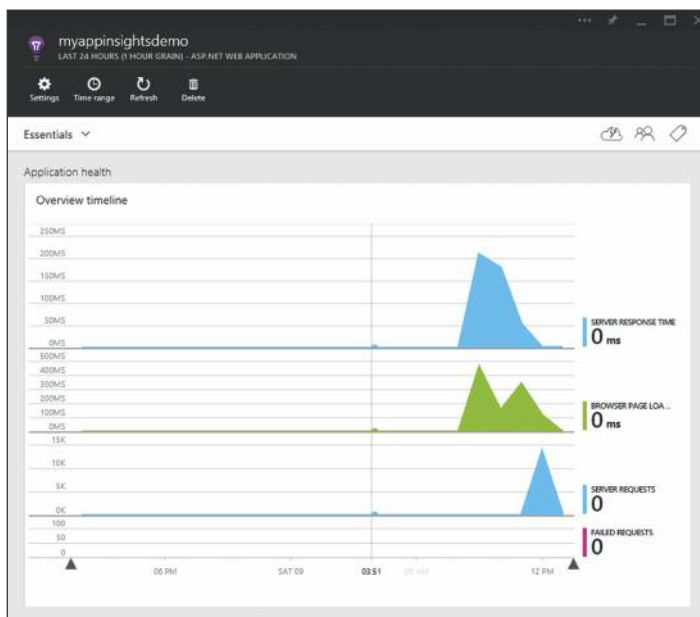


Figure 3 - First Performance Data in Dashboard

Looking at application diagnostics

With application diagnostics you can get valuable insights into why things might have gone wrong in your application. With Application Insights you can search right from the portal. It is a centralized location where you can search for the log events you send out and it can also correlate with the performance and usage metrics you have gathered. So when things go bad in your application this provides a complete and holistic view on what went on in the system, what was used at that moment and how things broke in your system. Without adding any line of code, Application insights will track any unhandled exceptions. If you want to send out specific log information with the optional levels like Verbose, Warning and Error you can do so by adding a two lines of code. Here is a code snippet on how to do this.

```

TelemetryClient client = new TelemetryClient();
client.TrackTrace("This is diagnostic information",
SeverityLevel.Warning);

```

If you go to the portal you can then find those log messages, and filter on anything that was happening surrounding that particular log message. This is shown in the following screenshot:

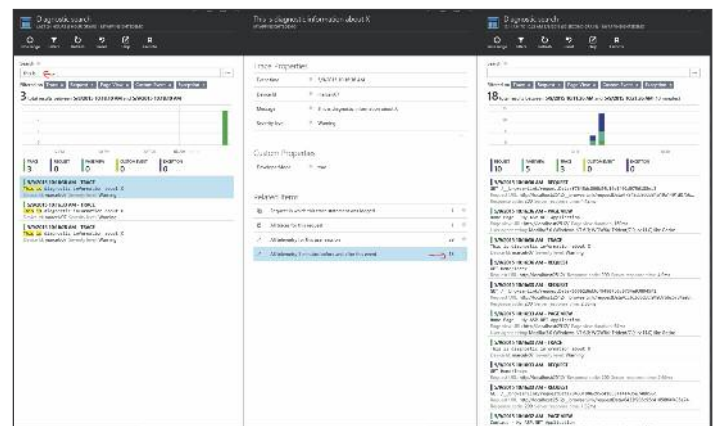


Figure 4 - Diagnostic Events

As you can see, finding specific diagnostics is something that can be done pretty easy by just adding these statements to your application

Measuring application usage

Before we look at how we can add usage metrics into our application, I want to take it one step back and talk a little bit about the theory behind why this is an important step into becoming more efficient in building software.

Eric Ries wrote in his book "The lean startup" about how to determine if you are delivering the right product. He describes a lot of principles on building a startup. One of the principles is the concept of testing assertions on what we think is most valuable for our customers. Traditionally we spend a lot of time specifying what a feature looks like and then we start implementing it. If we have continuous delivery set up we can even deliver it to production in a rapid pace. But the problem that remains with this approach is that the assumptions on why we build a certain feature are not tested and assumptions are made on the value of a feature before we can actually validate if this is true. We still tend to spend a lot of time and money delivering the wrong features to the customer when these assumptions are wrong. This all boils down to the simple principle that the only feature that provides value is a feature that is actually used and meets the end users need. To mitigate the risk of building the wrong things, Eric describes the concept of "validated learning", where you set up an experiment in your software and define the metrics to show you if the feature will bring to you what you assume. When it is established that the feature really provides value because the customer is using the feature as envisioned, only then do you start spending more time and money polishing that feature. He describes a so called Minimal Viable Product (MVP) that you build to enable you to test your assumptions on the feature you want to deliver. So you create a first simple iteration of the feature instead of spending a lot of time specifying the feature and then build it completely. You build the first minimal version to test your assumptions and after that you iterate on it to improve it with the help of the direct customer feedback.

With Application Insights you can build metrics into this Minimal Viable Product and learn if assumptions you made are true and based on that input pivot on your ideas and create a new experiment, or pursue the idea and polish it and measure the effects of those changes. In Application Insights this is done with telemetry events you can send and based on this you can build your own graphs to see the experiment and learn, or even export the data to some analysis environment of your own liking.

Let's have a look on how we can add this kind of telemetry data into your application.

Adding custom telemetry tracking to your application

With application insights you have the ability to add custom tracking to your application both client- and server-side. For this Application Insights provides the following tracking methods:

Method	Description
TrackPageView	Called when you show a page, a screen, a windows form, etc.
TrackEvent	Called to track User actions. It is used to track user behavior, so you use this to track e.g. if someone is actually using your new feature and you want to track that usage and the performance of those actions.
TrackMetric	Called to track a Performance measurements such as e.g. queue lengths or other metrics relevant for your application. You use this when it is not directly tied to a user action. Otherwise you use the TrackEvent and provide it also metric information in that call so it is directly correlated to the event.
TrackException	Called to log exceptions for diagnosis. It will trace where they occur in relation to other events and also provides stack trace information.
TrackRequest	Called to log the frequency and duration of server requests for performance analysis
TrackTrace	Called for tracking diagnostic log messages, as shown in the previous paragraph.

Table 1 - Tracking methods available on TelemetryClient

Now let's say you would like to set up an experiment where you want to validate if a customer will remove items from your shopping basket more often when they are from a specific region in the world. You also think it is related to the total amount of the Shopping Basket. You segment this amount into buckets of 0-100, 100-500 and 500-1000. To set up such an experiment, the only thing you need to do is add additional telemetry data to a call TrackEvent when someone hits the Delete button in your shopping cart page. This can be done with the following code snippet:

```
var basketValue = GetShoppingbasketTotalRange();
TelemetryClient client = new TelemetryClient();

var properties = new Dictionary<string, string>();
properties.Add("Amount segment",
    GetShoppingbasketTotalRangeSegment(basketValue).ToString());

var measurements = new Dictionary<string, double>();
measurements.Add("Amount total", basketValue);
```



```
client.TrackEvent("Item removed", properties, measurements);
```

In this code sample you can see we add custom properties to the tracked event. This gives us the ability to slice the data using those property values in the portal. We also add a measurement to the event, that is used to show the average value of the shopping basket, when the button is clicked. Since Application Insights already tracks from which country requests come from, you don't need to add this as a custom property for slicing your data. When you add this piece of code to your application, you can create a graph or table that shows this data in the portal. You can see the results in the following screenshot:

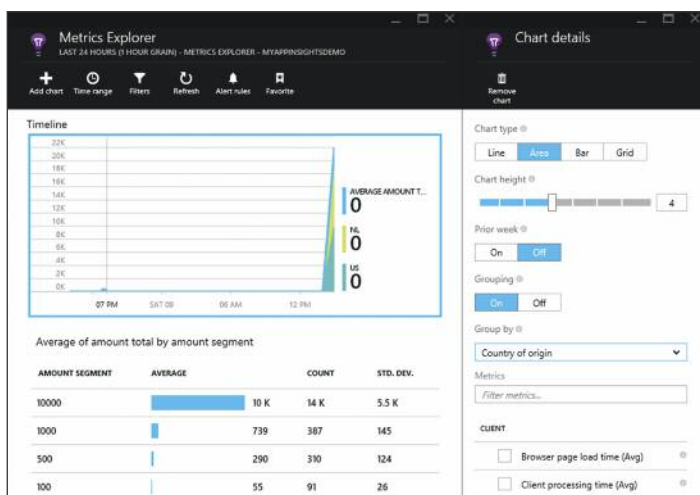


Figure 5 - Custom Telemetry Data Dashboard

Measuring application uptime

Finally, we can also track how well the uptime of your website is. This is done by configuring Application Insights inside the Azure portal to send a request to your website on a certain configurable interval. The most simplistic uptime test is that you provide an URL of your website, e.g. of your home page. When a request to this page returns the 200 OK return code, it is considered as your site being up. You can also define a more complex set of checks on your website to validate if everything runs as expected. For this you can either record a Fiddler trace or use Visual Studio web performance tests to record a set of page request and additional validations. Here you could validate if you can browse your e-commerce website and add an item to a shopping cart and go to the checkout to see if that flow is up and running. After recording these web requests at the Http level, you can upload this

recording into Application Insights. These tests are then configured to run from the Azure datacentre on the set interval.

When errors occur or test assertions fail, you can get email notifications and also dive deeper into the events that happened in that particular time window. This way you can diagnose why your site might have experienced an outage or was not able to return the pages within the set time window specified in the tests. The moment you see downtime in the portal, you can then dive into that specific event and dive deeper on the cause, by using all the things we discussed before. All this telemetry data is at hand, when you want to diagnose such an outage problem.

Conclusion

With Microsoft Application Insights it becomes possible to get a full view of your application running in production. Not only can you track performance, diagnostics and see the uptime of your website, you can also track very specific metrics about the usage of your product. By using all these metrics as input to your agile development process, you can ensure you are building the things that matter most to your customer. With Application Insights you are able to optimize your software development by implementing the full "Build, Measure, Learn" loop.



Marcel de Vries

CTO

<https://xpirit.com/specialists/marcel-de-vries>

mdevries@xpirit.com



An Introduction to Azure Service Fabric

Building hyper-scale distributed applications can be very complex. Developers need to take asynchronous communication, concurrency, latency, redundancy into consideration and many more aspects that are necessary to make distributed applications successful. These aspects are necessary, but they aren't directly related to the business domain the applications relate to. Microsoft Azure Service Fabric helps to simplify these aspects and have developers focus on the business domain at hand.

Next Generation PaaS

Microsoft started their Cloud Computing platform as a Platform-as-a-Service (PaaS) offering with Web- and Worker Roles. Web- and Worker Roles basically are templates for virtual machines that are managed entirely by Microsoft. They hold your applications code and they define the characteristics of the virtual machines that code is deployed to. Although you can deploy more than one applications onto a single virtual machine (or role instance), updating any of the applications can take up to 15 minutes and thus typically a role instance (or virtual machine) contains only a single application.

Web- and Worker Roles are completely stateless due to the fact that the actual disks are located on the physical machine where the role instances are hosted. To account for failure of a physical machine or software errors, Microsoft recycles our role instances. This means that the disks will be replaced with new disks and that our application code will be deployed on those new disks. Any state that was added after initial deployment will be lost. To make sure that our state is being saved, we need to store it in a durable remote data store. Examples of durable remote data store are Azure SQL Database, Azure Table- or Blob Storage, Azure DocumentDb, Azure Search, etc. Although these data store serve their purpose very well, you still need to

think of latency, concurrency, consistency and implementing retry policies. These are all things that complicate your overall solution without adding any business value. In addition, using these platform services makes it harder to build cross-premises or cross-cloud solutions. Azure Service Fabric provides a layer on top of the operating system of the virtual machine hiding details like the number of cores, the amount of RAM etc. This layer, or fabric as it's called, is created over a cluster of nodes (or virtual machines) and it manages the provisioning, redundancy, communication and all the complexity you don't want to deal with. Azure Service Fabric also provides a quorum-based replication mechanism that allows you to store data where your application code runs. This greatly reduces latency, while preserving state on multiple machines, potentially in multiple data centers. And the great thing is that, because Azure Service Fabric is implemented as a layer on top the operating system, it can run anywhere. As long as nodes can find each other at IP level they can be joined to the cluster.

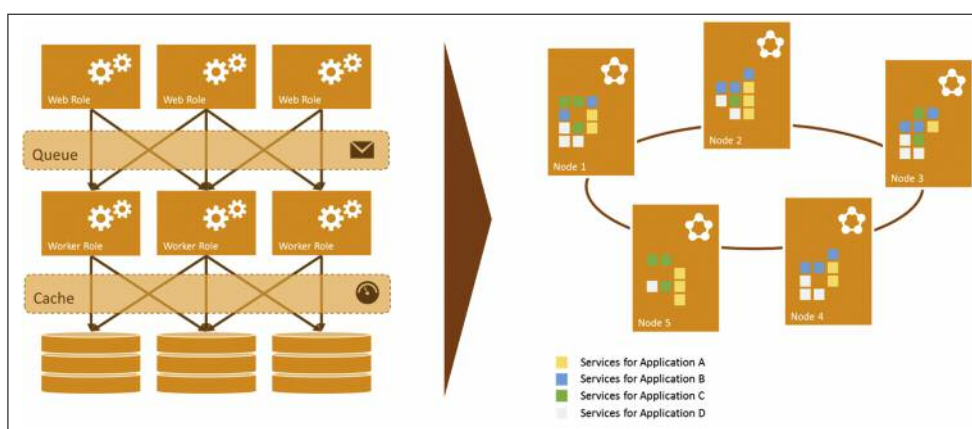


Figure 1 - Reduced Complexity

Getting started with Azure Service Fabric

The first thing you will need to do is setup your development environment. To get started first install Microsoft Azure Service Fabric SDK – Preview 1 using the Web Platform Installer. For this you need to have Visual Studio 2015 RC installed.

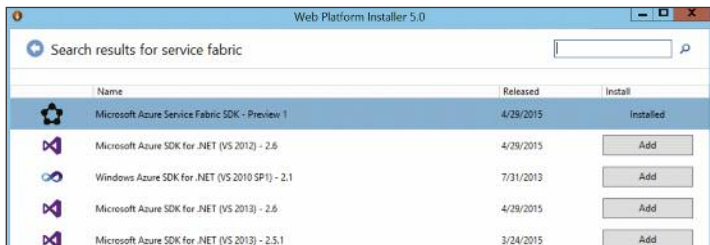


Figure 2 - Web Platform Installer

This will install the Microsoft Service Fabric Host Service, the Service Fabric SDK, the Microsoft.ServiceFabric.Powershell module and the Microsoft Azure Service Fabric Tools for Visual Studio 2015. Next you'll need to set up a development cluster to test your code on. A Service Fabric development cluster is very different from the Compute Emulator we all know from Web- and Worker Roles. A Service Fabric development cluster is exactly the same technology that runs in your production environment ensuring that the code that runs in your machine will run exactly the same way in your production environment. The only difference is that the cluster nodes in a development cluster are running on a single machine whereas in production each node runs on a separate machine. These machines can either be physical or virtual, on-premises, in your ISPs datacenter, in Microsoft's datacenter (Azure) or in any Cloud environment for that matter, that runs Windows Server 2012 R2 and up.

Creating a development cluster is done in four simple steps:

1. Open up a PowerShell window as Administrator
2. Execute `Set-ExecutionPolicy -ExecutionPolicy Unrestricted -Force -Scope CurrentUser`
3. Execute `cd "$env:ProgramFiles\Microsoft SDKs\Service Fabric\ClusterSetup"`
4. Execute `.\DevClusterSetup.ps1`

This will start the Microsoft Service Fabric Host Service and it will setup a 5 node cluster on your machine. By default your cluster files (logs and data) will live in `c:\SfDevCluster`.

Finally, you can verify your cluster by starting the Service Fabric

Explorer located in the Tools folder in the Service Fabric SDK installation directory (`C:\Program Files\Microsoft SDKs\Service Fabric`).



Figure 3 - Service Fabric Explorer

Building Microservices

Microsoft Azure Service Fabric is based on a microservices principles. Applications that run on Azure Service Fabric consist of small autonomous services that interact with each other in a loosely coupled manner. These services can be deployed together as one application, but they can be upgraded separately without causing downtime to the system.

Reliable Services Api

To get started you'll need to create a new project in Visual Studio using one of the templates in Visual C# -> Cloud -> Service Fabric. There are four templates, two for building on top of the Service Model API and two for building on top of the Actor Model API (which will be discussed later). The two templates for building on top of the Service Model API are:

1. Application with Stateless Service
2. Application with Stateful Service

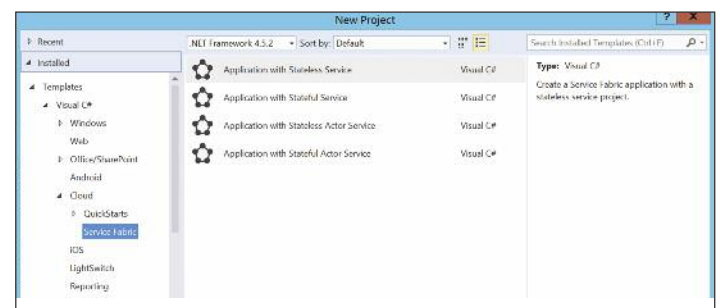


Figure 4 - Visual Studio Project Templates for Azure Service Fabric

As the name suggests the templates consist of one project for the application and one project for a service. You can look at the application as a configuration container for a number of services. Configuration is done through an `ApplicationManifest.xml` file. In there you can specify endpoint names that are used for services, the minimum number of replicas that should be acknowledged before returning and the target amount of replicas that should exist for a particular service and the required partitioning strategy, etc. The other project is the actual service. From a microservices perspective this should be a small autonomous service with a single responsibility. Configuration and definition of the service is done through a `ServiceManifest.xml` file, which is located in a `PackageRoot` folder.

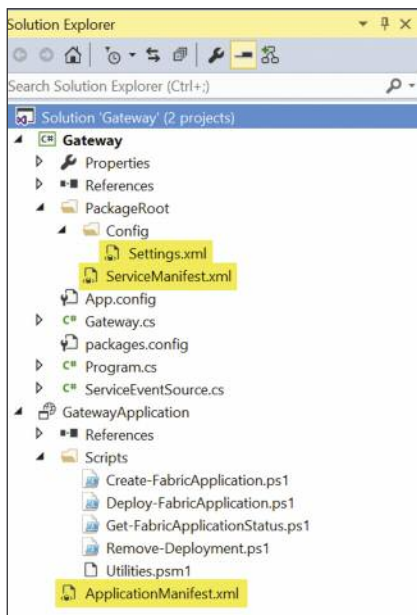


Figure 5 - Service Fabric Application

Stateful services, on the other hand, store their state on the actual node where they're being hosted on. This reduces network latency while retrieving the data. Partitioning, allocation and replication are all managed by Service Fabric based on the configuration provided in both the `ApplicationManifest.xml` and `ServiceManifest.xml` files. State will be stored on the actual node where your service is hosted, thus reducing network latency.

Once you created an application, additional services can be added to the application by right-clicking the applications project in the Solution Explorer. This way you can mix all types of services (Stateless, Stateful, Stateless Actor and Stateful Actor) in the same application. The `ApplicationManifest.xml` file will automatically

be updated to include the new services and you can manually alter them to override configuration settings made in the `ServiceManifest.xml` file.

Reliable Services

Writing a service can start out simple, but you still have the option to change the inner workings of a service.

A Service Project basically is a Console Application that compiles to an EXE. In the `Main` method a `FabricRuntime` is created and a service type is registered. Multiple service types can be registered to the `FabricRuntime`, but that would mean these service types can only be upgraded together and not separately. We'll discuss this later when we talk about the Actor Model API.

There are two types of services, stateless and stateful. Stateless services do not contain state on the nodes that they're hosted on. They can have state in external data stores like Azure SQL Database, Azure DocumentDB (or any other storage service available), but their state will not be stored on the actual node.

```
using (FabricRuntime fabricRuntime = FabricRuntime.Create())
{
    // This is the name of the ServiceType that is registered with FabricRuntime.
    // This name must match the name defined in the ServiceManifest. If you change
    // this name, please change the name of the ServiceType in the ServiceManifest.
    fabricRuntime.RegisterServiceType("GatewayType", typeof(Gateway));

    ServiceEventSource.Current.ServiceTypeRegistered(Process.GetCurrentProcess().Id, typeof(Gateway).Name);

    Thread.Sleep(Timeout.Infinite);
}
```

Snippet 1 - Registering Service Types to the FabricRuntime

A second class in the project represents the actual service. The service class inherits either from `StatelessService` or `StatefulService` and it requires you to implement just two methods:

1. `CreateCommunicationListener`
2. `RunAsync`

```
2 references
public class Gateway : StatelessService
{
    1 reference
    protected override ICommunicationListener CreateCommunicationListener()
    {
        // TODO: Replace this with an ICommunicationListener implementation if your service needs to handle user requests.
        return base.CreateCommunicationListener();
    }

    0 references
    protected override async Task RunAsync(CancellationToken cancellationToken)
    {
        // TODO: Replace the following with your own logic.

        int iterations = 0;
        while (!cancellationToken.IsCancellationRequested)
        {
            ServiceEventSource.Current.ServiceMessage(this, "Working-{0}", iterations++);
            await Task.Delay(TimeSpan.FromSeconds(1), cancellationToken);
        }
    }
}
```

Snippet 2 - Two overrides of a Service

`CreateCommunicationListener` needs to return an implementation of `ICommunicationListener`, which deals with opening and closing a communication channel and listening on it. This can be a communication channel of your choosing (e.g. WebSockets, HTTP, Azure Service Bus, ZeroMQ, etc.). This communication channel

is used to receive messages from outside the service.

RunAsync is where all the work is done. If you're familiar with how Worker Roles work, this is very much the same as the **Run** method in a Worker Role. It defines the lifecycle of the service. If the **RunAsync** method returns the service will be recycled. So, just as with Worker Roles, it contains an endless loop that will execute our code.

Reliable Data

If you chose to create a Stateful Service your service class will contain a **StateManager** property. This property can be used to get reliable data collections, queues and transactions. This way you can read from and write to a collection as you are used to and you don't have to worry about concurrency, redundancy, availability or scale.

```
0 references
protected override async Task RunAsync(CancellationToken cancellationToken)
{
    // TODO: Replace the following with your own logic.
    var myDictionary = await this.StateManager.GetOrAddAsync<IReliableDictionary<string, long>>("myDictionary");
    while (!cancellationToken.IsCancellationRequested)
    {
        using (var tx = this.StateManager.CreateTransaction())
        {
            var result = await myDictionary.TryGetValueAsync(tx, "Counter-1");
            ServiceEventSource.Current.ServiceMessage(
                this,
                "Current Counter Value: {0}",
                result.HasValue ? result.Value.ToString() : "Value does not exist.");
            await myDictionary.AddOrUpdateAsync(tx, "Counter-1", 0, (k, v) => ++v);
        }
        await tx.CommitAsync();
    }
    await Task.Delay(TimeSpan.FromSeconds(1), cancellationToken);
}
```

Snippet 3 - Using Reliable Collections

A Programming Model Built For Microservices

So microservices are small autonomous services with a single responsibility. Now that looks a lot like Separation of Concerns in object-oriented programming. Basically each class would be a microservice on itself if it was independent from the process it ran in. Project Orleans was developed to create a programming model that implements the mathematical Actor model for concurrent computations. This programming model was used to accommodate the hundreds of thousands of game sessions and play records of the computer game HALO 4.

The Reliable Actors API in Azure Service Fabric is a programming model based on Orleans built on top of the Reliable Services API.

A Short Note On Virtual Actors

Before we continue, here's a short note on what virtual Actors are. Virtual Actors are isolated single-threaded components that

encapsulate both state and behavior. Actors interact with the system, including other Actors, by sending asynchronous messages in a request-response pattern. Virtual Actors are always there. If they are not, they will be created on the fly and if they were created in the past they will exist for ever. So the developer is not concerned about the lifecycle of the actual object.

```
var game = ActorProxy.Create<IGame>(new ActorId("game::12637674829"), ApplicationName);
var gamePlayerCount = await game.GetCountAsync();
```

Snippet 4 - Creating an Actor

Reliable Actors Api

Up until now we haven't discussed the other two project templates, Application with Stateless Actor Service and Application with Stateful Actor Service. The structure of the application is pretty much the same in that it still is a configuration container for a collection of services. The Actor Service templates both create two projects in addition to the application project:

1. A project for the actual service
2. A project that contains the interfaces for the Actors

The actual service host is exactly the same as with the Reliable Services API. The only difference is that we need to register every Actor to the FabricRuntime. Every Actor will become a Service in Azure Service Fabric. And because the Actor Service by default is configured to have nine partitions, activations of the Actors will be distributed over the nodes in the cluster automatically.

```
using (FabricRuntime fabricRuntime = FabricRuntime.Create())
{
    fabricRuntime.RegisterActor<IGame>();
    fabricRuntime.RegisterActor<IPlayer>();

    Thread.Sleep(Timeout.Infinite);
}
```

Snippet 5 - Registering Actors to the FabricRuntime

```
1 reference
public interface IGame : IActor
{
    1 reference
    Task<int> GetCountAsync();

    1 reference
    Task SetCountAsync(int count);
}
```

Snippet 6 - Implementing IActor

Every Actor starts out as an interface defining its behavior. This interface should implement the **IActor** interface. The **IActor** interface is just a marker interface for the **ActorProxy** to validate that this will indeed return an Actor.

Next, we need to create a class in the actual Service project that inherits from **Actor** and implements the previously created interface. If it involves a Stateful Actor Service we'll need to inherit from a generic **Actor** of the type of the state it will contain. The state needs to be a class that needs to be data contract serializable.

```
[DataContract]
2 references
public class GameState
{
    [DataMember]
    public int Count;

    14 references
    public override string ToString()
    {
        return string.Format(CultureInfo.InvariantCulture, "GameState[Count = {0}]", Count);
    }
}
```

Snippet 7 - Actor State object

```
1 reference
public class Game : Actor<GameState>, IGame
{
    0 references
    public override Task OnActivateAsync()
    {
        if (this.State == null)
        {
            this.State = new GameState() { Count = 0 };
        }

        ActorEventSource.Current.ActorMessage(this, "State initialized to {0}", this.State);
        return Task.FromResult(true);
    }

    1 reference
    public Task<int> GetCountAsync()
    {
        ActorEventSource.Current.ActorMessage(this, "Getting current count value as {0}", this.State.Count);
        return Task.FromResult(this.State.Count);
    }

    1 reference
    public Task SetCountAsync(int count)
    {
        ActorEventSource.Current.ActorMessage(this, "Setting current count of value to {0}", count);
        this.State.Count = count;
        return Task.FromResult(true);
    }
}
```

Snippet 8 - An Example of a Game Actor

Snippet 8 shows that Stateful Actors have a State property where state can be stored. Persisting the state of an Actor is completely transparent and you can only specify whether to store the data on disk or in-memory (by using the **VolatileActorStateProviderAttribute** above the Actor class).

If an Actor is idle for more than sixty minutes it will be deactivated and ultimately it will be garbage collected. If you need to trigger a specific method on an Actor periodically, you can register

so-called Reminders. Reminders are registered with and controlled by the FabricRuntime and thus they will always fire, even when the Actor is deactivated or even garbage collected.

Just like Grains in Orleans, Azure Service Fabric Actors can raise events to which other actors or client services can subscribe.

Conclusion

Microsoft Azure Service Fabric offers developers a virtual layer, on top of the operating system, that spans multiple machines. This layer helps orchestrate our applications keeping them healthy even when the physical machine is failing. Because Azure Service Fabric takes a Microservices approach, services that make up an application can be upgraded independently from each other "without downtime and within seconds. And because Azure Service Fabric is a layer on top of the operating system it can run anywhere. It hides all the complexity that comes with developing highly available, high performance distributed systems and lets developers focus on the business domain at hand.



Patriek van Dorp

Lead Consultant

<https://xpirt.com/specialists/patriek-van-dorp>



Develop Cross-platform mobile hybrid apps with Ionic

Before you start with the development of a mobile app, there are some tough choices to be made. The most important one is how the app will be developed. You have the option to develop it as a native, HTML5 or a hybrid app. If you choose to develop a native app, the number of mobile platforms that you would like to support usually states the number of apps that will be developed.

Currently you probably want to support at least iOS and Android, perhaps Windows Phone. The consequences are you will develop in three different environments, in three different languages and the apps need to be developed in three different development tools. It's clear a lot of challenges needs to be addressed. The main challenges are you will need a broad technical knowledge and it will be hard to keep the apps functionally equal.

One of the available solutions is Xamarin. Xamarin allows you to develop a native app in one language, C#, for all platforms. Especially with Xamarin Forms you have only one codebase. However, Xamarin Forms is still maturing, since it's a relatively new product. Luckily, Xamarin becomes more stable with each update. Some of Xamarin's major advantages are quick app response, platform specific look and feel and the use of XAML to define the views.

If you have experience with XAML, you will quickly feel familiar with Xamarin Forms.

An alternative to these native apps is a HTML 5 solution. Basically you develop a web application which will be used and themed as an app. Just

like Xamarin there is a single code base, used for all platforms. When the development team has experience in web development, this approach is probably the most appropriate. The drawback to this solution is there are restrictions with respect to access device hardware. For example, you cannot use the camera or the GPS. Since HTML rendering is browser specific it can be a challenge to make your app look and feel exactly the same on each platform. Another disadvantage is there is always an Internet connection required, thus data loss could occur.

For a long time it's possible to opt for a hybrid solution. This is an HTML5 app, hosted in a native container. Examples of such containers are PhoneGap, Cordova and Trigger.io. When you choose for a hybrid app, you choose for both the benefits of an HTML5 and a native app. The native container has access to device-specific functionality and removes the internet connection availability requirement. This solution has some disadvantages: the end user might experience your app as slow and the look and feel is different from what the end user of a native app is used to.



Hybrid apps 2.0

As in the entire software world, the world of hybrid apps is moving fast. Last year many frameworks popped up to provide a solution to the classic disadvantages of a hybrid app with HTML5. There are Famo.us, Framework7, Monaca, AppGyver, OnsenUI and Ionic. The latter looks promising, with Ionic, apps can be developed rapidly and cross platform. When you have basic experience with webdevelopment, in particular AngularJS, Ionic is the next logical step.

Ionic

Ionic is an open source framework for building HTML5 mobile apps to give the end user the experience of a native app. This is done by making use of HTML, JavaScript and CSS. Like a number of other frameworks AngularJS is used for the User Interface. Ionic exists for almost a year, after a long list of betas and release candidates, Ionic is about to release version 1. Ionic is already mature enough for developers to put their applications into production.



Below a list with some need-to-knows about Ionic:

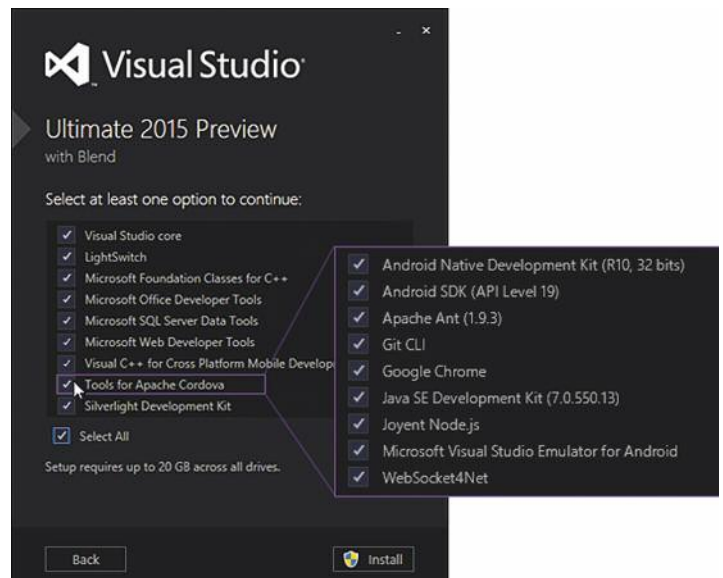
- Ionic works on Ionic 2, which supports Angular 2
- Optimized for speed
- On both iOS and Android, apps look like a native app. An end user will not be able to quickly tell whether the app is native or hybrid
- Windows Phone is not supported (yet). The support of this is on the roadmap
- All native app containers are supported, but Cordova is favored
- A major advantage is the existence of a large community. There is a high response rate and speed on the forum
- Ionic is open source. The license even allows commercial apps to be developed without the app being open-source itself.
- Drifty just invested 2.6 million dollars
- The book 'Developing an Ionic Edge' has just been released
- The book 'Ionic in action' will be released soon
- Ionic provides more and more services and tools to simplify the app-developers life
- Ionic is 100% free to use
- There are numerous apps in the stores developed with Ionic. For example, the Android and iOS apps Sworkit and Chief Steps are a good example to see there is almost no difference from a native experience
- Ionic focuses on the latest versions of iOS and Android. At this time, iOS 6 and Android 4.1 and later are supported. Ionic is working hard to become a mature framework to develop cross-platform mobile application with.

Starting with Ionic

To install all necessary software to be able to start developing, seems to be a project in itself, but it's well documented by Ionic (<http://ionicframework.com/getting-started/>).



Ionic requires NodeJS and Apache Cordova. Besides this also the Android SDK and Java SE Development Kit need to be installed. There are several ways to install all necessary software. Of course it's possible to follow Ionic's documentation. Make sure that the correct System Paths are set in Windows. Another way is particularly useful if you want to use Visual Studio as a development tool. You can limit the actions by only installing or configuring Visual Studio Tools for Apache Cordova. For Visual Studio 2013 with update 4 the installer can be found here: <https://www.microsoft.com/en-us/download/details.aspx?id=42675>. For Visual Studio 2015 it's a matter of checking the correct checkbox in the setup.



After installation, a new project template is available in Visual Studio. The new template is called 'Blank App (Apache Cordova)' and can be found under Templates -> JavaScript -> Apache Cordova Apps.

The most common way to develop apps with Ionic is often done with Sublime Text, a free third-party tool. Visual Studio is also a viable option.



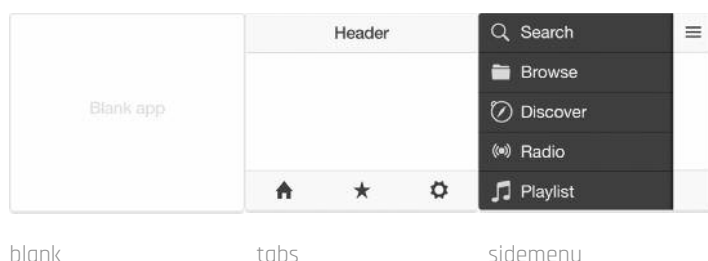
Ionic offers an alternative to start developing. For this, the project Ionic Box is available. It's a virtual machine which can be downloaded with all necessary tooling. More on this can be found at: <https://github.com/driftyco/ionic-box>

Create an app with Ionic

It is advisable to stay as close as possible to Ionic's intended manner to start a project. There are Visual Studio templates and Nuget packages available. However, it is important to know what these libraries actually do for you.

Ionic offers multiple templates to start developing an app:

blank	An empty app
tabs (default)	A starter app where tabs are used for navigation
sidemenu	A starter app where a sidemenu is used for navigation
Maps	A sidemenu starter app with Google Maps integration
Salesforce	Uses Salesforce OAuth authentication and the Salesforce REST API to call Salesforce functionality.
Analytics	A starter app which uses the Ionic IO Analytics Service
Push	A starter app which uses the Ionic IO Push Service
io	A starter app which uses all Ionic IO services



Start the NodeJS command prompt, navigate to the directory where the app should be created and type the following command to use the sidemenu template for a new app with the name 'my-

appname':

```
ionic start myappname sidemenu
```

A directory structure is generated with all necessary files inside. The www directory contains all files providing the app's user interface.

Run the project

There are several ways to run a project. Because an Ionic project is a web project, it can be run in the browser. In addition, it can run in an iOS or Android emulator. It's also possible to run the app in the device's browser or of course as a native app on a device. Since a device's browser interprets the app differently from the native app, it's advised not to use this option.

Run in the browser

To do this, execute the following command in NodeJS:

```
ionic serve
```

A local development server is started, the default browser launches and displays the project. A very productive feature is called LiveReload. This ensures that as soon as changes have been made to the source, the browser automatically refreshes. All changes are immediately visible without rebuild time, redeploy or local development server restart. Especially when you have a development environment with two screens, this is a relief.

Run in the emulator

This sample only shows how to start the emulator for Android. When you want to start the simulator for iOS, you will need a Mac. Execute the following commands:

```
ionic platform add android
```

```
ionic build android
```

```
ionic emulate android
```

The emulator is started and the app will be displayed.

Run as an app on a device

Before release it's necessary to test the app on an actual device. If the app should be installed on iOS, an Apple Developer license is required which costs \$99,- per year. After this XCode should be installed and configured. This is necessary because this way Cordova can package and sign the app for iOS. XCode can only run on Mac hardware. For Android, the following command can be executed in NodeJS after the Android device is attached to your PC. (USB debugging must be enabled on the device)



ionic run android

The project structure created by Ionic is very well arranged. The project includes package manager Bower. Ionic is already loaded with Bower and can be used immediately to load other packages.

Use Visual Studio to develop Ionic projects

Besides Sublime Text, Visual Studio can also be used to develop the Ionic project. This requires some manual steps. It is convenient to create a 'Blank App (Apache Cordova)' with the name www. Both the Ionic directory and the directory Visual Studio created for the project correspond. Close the solution and copy the sln file and all files and directories, except the images folder and index.html to Ionic's www folder. Open the solution from its new path and add all directories of Ionic to the project. Now both Sublime Text and Visual Studio can be used for development.

Ionic Components

Ionic offers a comprehensive set of components that can be used directly in the views. These are all well documented on <http://ionicframework.com/docs/components/>. Examples include List, Grid, Toggle, Cards, Checkbox, Button and many more components.

Additionally Ionic provides a layout for standard HTML tags such as h1 and p. No layout-specific code needs to be written for iOS or Android. Ionic is not just a CSS framework it is also a JavaScript UI library.

For example, JavaScript provides services to display a pop-up or a spinner. All services can be found here: <http://ionicframework.com/docs/api/>. This altogether provides a broad range of functionality to the developer, for conveniently developing the User Interface. Besides User Interface services, Ionic also offers other services.

Ionic ecosystem

Although Ionic only exists little over a year, the project is already in the top 50 most popular frameworks on GitHub. Ionic releases more and more tooling to make developers' life easier. The goal of Ionic is to give hybrid app developers a head start over developers of native apps.

Ionic IO

Ionic offers all services under the name Ionic IO. To use these services you need to personally register and the app must also be registered. After this, the following services are available:

Ionic User

This service provides user tracking. In addition, this service is required when using other services like Analytics and Push. <http://docs.ionic.io/v1.0/docs/push-quick-start>

Ionic Push

This service can send push notifications to the app. <http://docs.ionic.io/v1.0/docs/push-quick-start>

Ionic Deploy

Usually when updating your app, you have to go through the resubmission process. With Ionic Deploy new versions of the app can be deployed without going through the resubmission process. This type of update does not support binaries. Only HTML, CSS and JavaScript can be updated this way. <http://docs.ionic.io/v1.0/docs/deploy-install>

Ionic Analytics

This Ionic IO service provides a way to add Analytics, metrics and error tracking to your app. Of course you are free to choose a different analytics provider. <http://docs.ionic.io/v1.0/docs/analytics-install>

Ionic View

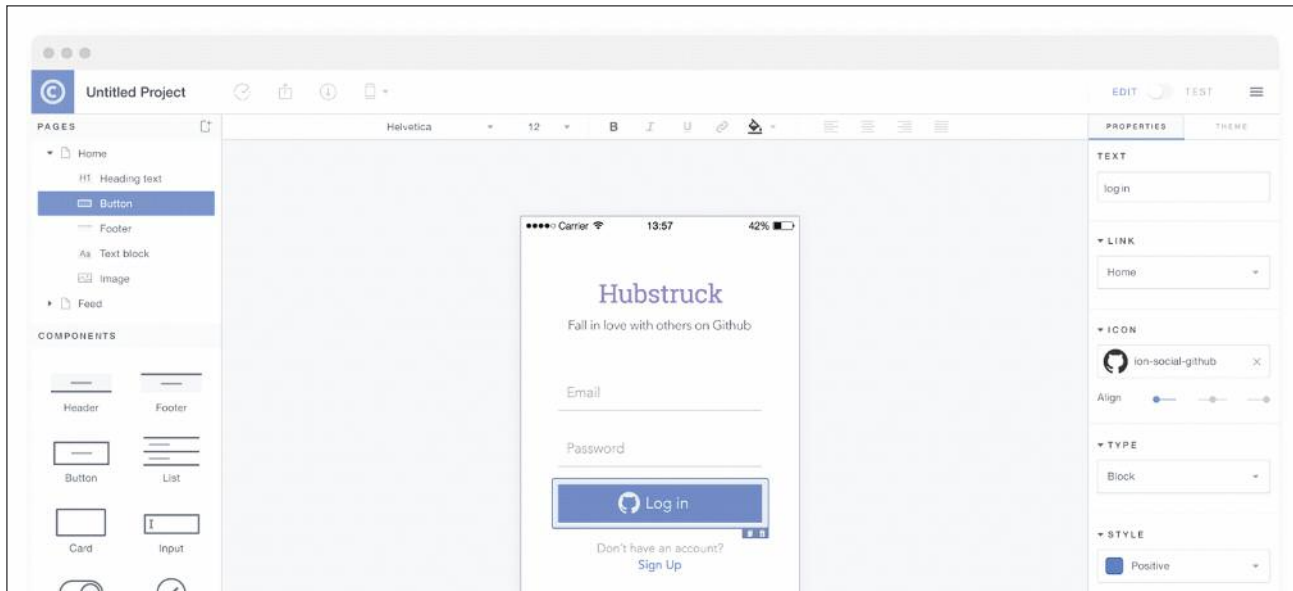
All people involved in the app development can install Ionic View on their device. With Ionic View they get access to the app under development and are able to review it. Ionic View is available for both iOS and Android. It's possible to upload your app via the command prompt and make it available. Ionic view is currently in beta. More on this at: <http://view.ionic.io/>

Icons and splashscreen

Every platform has different requirements regarding the images being used. This has to do with screen resolutions. Ionic takes away this burden by generating correct images, per platform, from a single source-image. Ionic does the same for Splashscreen images. More on this can be read at: <http://ionicframework.com/docs/cli/icon-splashscreen.html>

Creator

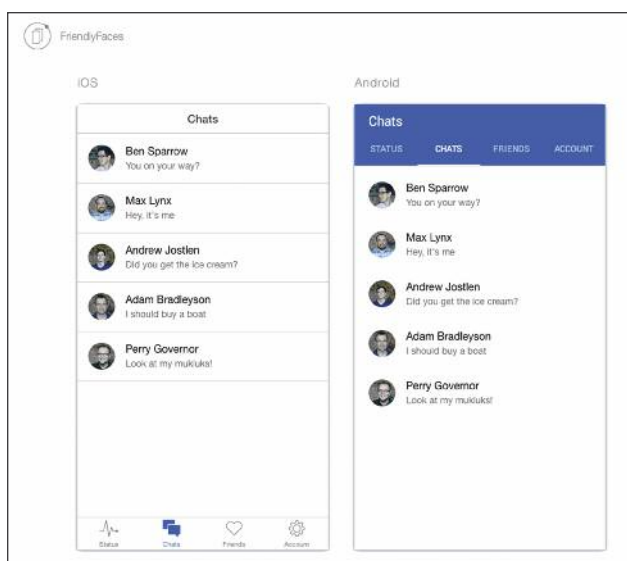
With Ionic Creator it's possible to create mockups and prototypes of mobile apps. After this a template can be generated which can be used to create an app. This ensures a quick start with the development of an app. The tool is not available yet. For more information: <http://ionicframework.com/creator/>



Ionic lab

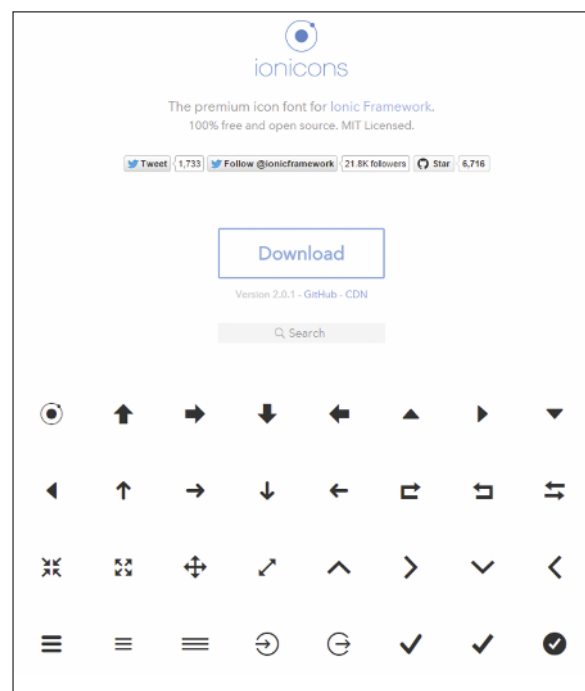
Ionic Lab displays the apps UI for both iOS and Android at the same time in a browser window. LiveReload is still working when using Ionic Lab. Both windows are being displayed side-by-side. To use this feature execute the following command:

```
ionic serve --lab
```



Icons

Over 700 icons, created by Ionic, can be downloaded and used for free. <http://ionicons.com/>



ngCordova

Ionic has created AngularJS wrappers to call native functionality on phones through Cordova. This functionality is grouped in a library called ngCordova. These wrappers work on both Cordova and PhoneGap. With this library functionality is made available such as taking a picture, a barcode scanner, upload a file, the flashlight, GPS and much more. ngCordova works on all AngularJS frameworks that operate on Cordova. <http://ngcordova.com/>



ngCordova

Conclusion

The world of mobile web development is very dynamic at this time. It is not yet clear which method of developing a cross-platform app will become the de facto standard. AngularJS is very popular among web developers and Ionic provides the ability to use AngularJS for cross-platform mobile apps. This is offered with powerful services that make it very easy for the developer to quickly develop an app with a native user experience.

Ionic is fairly new player and there are many competing frameworks in this area. Windows Phone is not yet supported. Ionic only supports recent versions of the mobile platforms. Especially because phones have to be fast enough to do the rather heavy rendering. In spite of this, in several cases such as long lists, apps might not feel like a native app. Ionic should give you the look and feel of a native app. My experience is that Ionic particularly delivers the looks but does not always give the feeling of a native app.

Ionic is working hard to resolve these performance issues. Maybe the performance problem can be found in the use of AngularJS. As soon as Angular 2 is released, Ionic 2 will be released short after.

Despite the drawbacks, Ionic has been embraced by many developers and has a large community. Without doubt this has to do with the fact that Ionic is open-source and free to use, even for commercial applications. The tooling that Ionic offers is very result oriented and reduces the choices a developer needs to

make, although he surely can. The LiveReload feature is really great and saves lots of time. All this could make Ionic a player in the mobile apps market that's here to stay. In addition, it's clear that hybrid apps are a formidable competitor to native apps and other cross-platform solutions like Xamarin.



Pascal Naber

Lead Developer

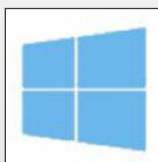
<https://xpirit.com/specialists/pascal-naber>

Are you ready for a next step in your Career?

Through experience you have an adventurous mind and now is the right moment in your career to be part of something new, something excellent and most of all something inspiring and fun. We ask for your energy, enthusiasm and devotion, nothing more, nothing less. Trust us, we will not disappoint you in what you get in return.



For our expanding assignments in the Enterprise Mobility space we are searching for a Mobile Lead Consultant with a firm knowledge of the Xamarin tools. For more details visit <https://xpirit.com/mobile-lead-consultant>.



We are searching for Microsoft Azure specialists and professionals with a firm knowledge of Microsoft Cloud services and tools. For more details visit <https://xpirit.com/cloud-lead-consultant>

Git vs Team Foundation Version Control: Getting started

When you are a Microsoft developer you probably are used to a Microsoft Version Control system: In the early days Visual SourceSafe or nowadays Team Foundation Server Version Control (TFVC). TFVC is fairly simple. Once you understand that the server always must know what you are doing it is simply a great and easy to use Version Control System.

When TFS 2012 introduced Local Workspaces, it became even better, because now all the advantages of TFS and the advantage of not having to check-out files, like in for example SVN, were combined. With a local workspace you do not have to inform the server about everything you are doing, you only check-in files. All changes are kept locally on your disk. This change in TFS simplified it even more.

But Microsoft realized that their Source Control System was not embraced by anyone other than developers using Microsoft tools. Within this cross-platform, cross-language world they need to have something different to offer. The best choice Microsoft made was not to build something themselves, but embrace a system that was already very popular within the open source and non-Microsoft community: Git.

There are many great posts and blogs about what Git is. The book Git Pro, which is available online, does a great job in explaining the concepts. The most important thing to know is that Git is a Distributed Version Control System (DVCS) where TFS is a Centralized Version Control System (CVCS). TFS knows about all the changes that happen. You check out a file, edit the file and check it in. The server knows. History is on the server. Therefore you need to be "online" in order to use a Centralized Version Control System. A DVCS is different. You get a full copy of a repository that is stored somewhere on a server, including all the history etc. Then you work locally on this repository. You can check-in changes, view history and rollback if you like. After working for a while you can then push back all the changes to the repository where you originated from.

Because it can be very hard to grasp Git as a hardcore TFVC user this article contains some starting points to "translate" Git to TFVC.

Be aware that TFVC and Git are conceptually very different and that a "real" translation can actually not be made. But it will certainly help in setting some reference points.

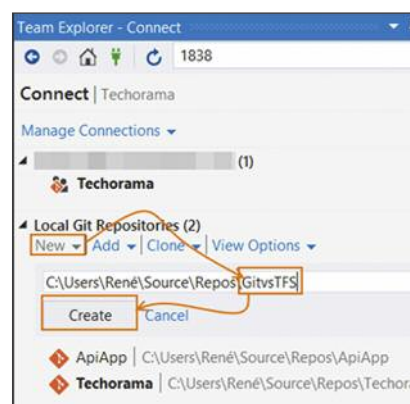
Dictionary

In this article we will talk about the following definitions:

Git Actions	TFS Command
Clone	Create Workspace and Get Latest
Checkout	Switch workspace / branch
Commit	CheckIn / Shelve
Status	Pending Changes
Push	CheckIn
Pull	Get Latest Version
Sync (Push and Pull, Only exists in VS UI)	CheckIn and Get Latest Version

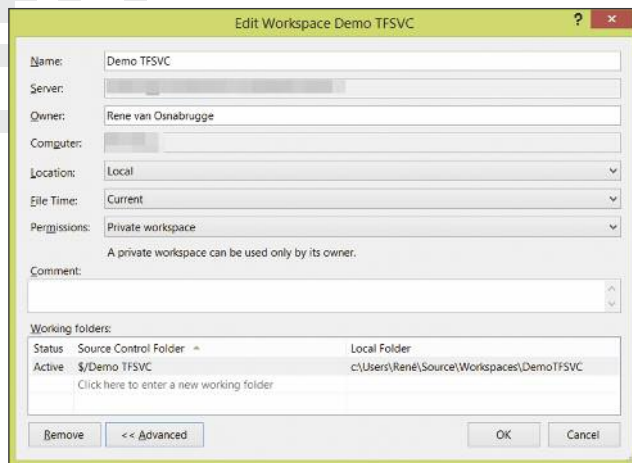
Clone vs. Create Workspace and Get Latest

[Clone] in Git means that you get a local copy of a full Version Controlled repository that is stored somewhere.



Create a repository clone in Git

You basically pull everything on to your local machine. Clone is the first thing you need to do when you start working with a Git repository.

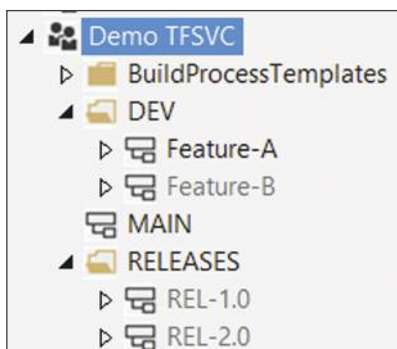


Create a workspace in TFS Version Control

In TFS we first [create a workspace], map the server folder to a local folder and [Get the Latest Version] of all the files in our workspace.

Checkout vs. Switch workspace / branch

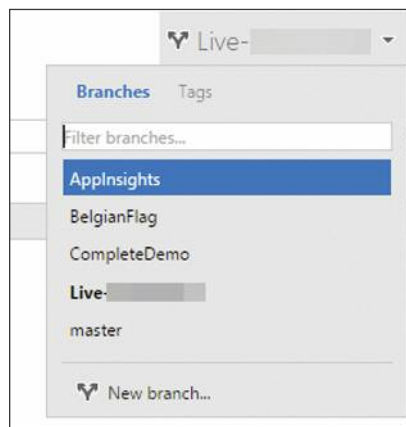
[Checkout] in Git, means change the branch you are working on. This is something that we do not have in TFS. Sure, we have branches in TFS, but they are in separate folders and locations on disk and in the repository.



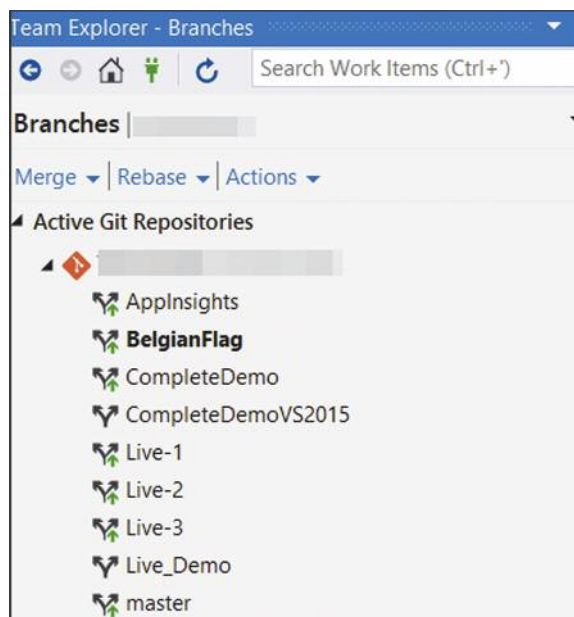
Branches in TFS Version Control

In Git this is not the case. You just have 1 view on the repository and the branches are "contained" within this repository. Switching branches is very easy to do and files directly change.

In TFS a best practice is, that you create a separate workspace per branch you are working on. When you want to work on another branch, you [change your workspace], open the solution from this workspace and start working. In Git you can use the command [Git Checkout] or just use the DropDown lists in the UI.



Switch Git branch in TFS Web Access

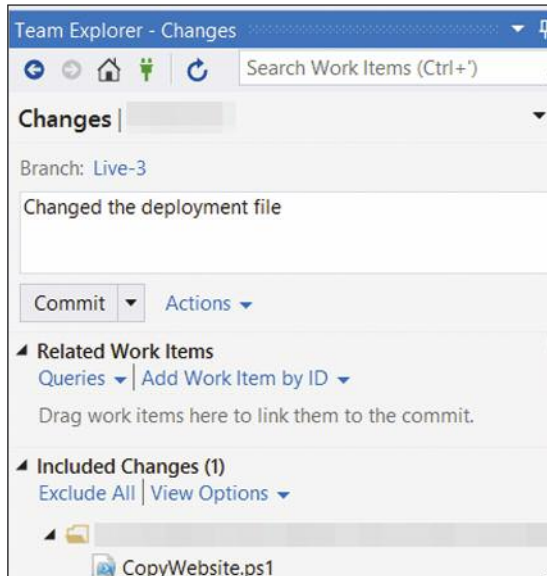


Switch Git branch in Visual Studio

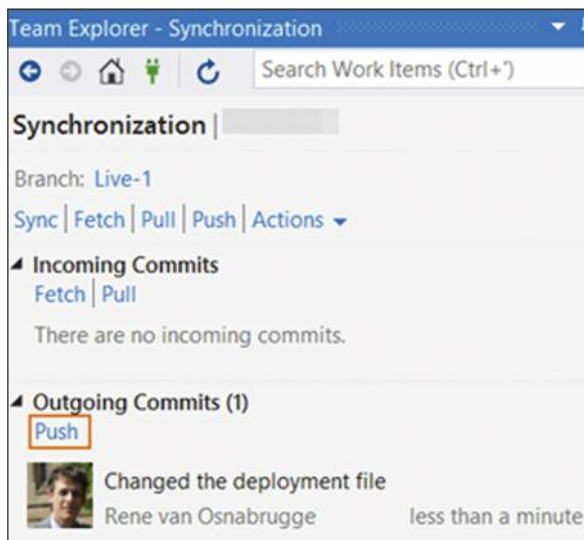
Commit vs. Checkin / Shelve

The comparison between commit and checkin/shelve is one that can actually be not made. In TFS you can perform a [Checkin]. The client sends all the files to the central TFS repository and the files are available to everyone. In Git you can [Commit] your changes when finished. Big difference here is that in Git, you always [Commit] to your "local" repository.

Changes are not available on the original branch where you came from. You have to [Push] your changes as a second action to “check-in” on the server. In TFVC you cannot commit locally. The closest thing to checking in for yourself is a shelve set.



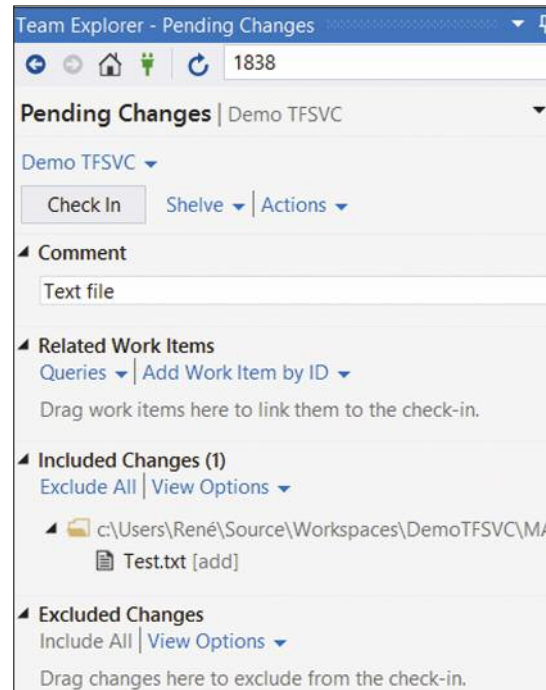
Git Commit



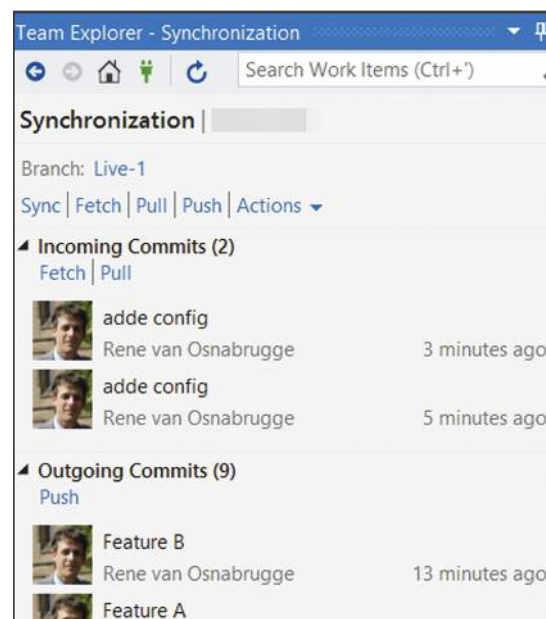
Git Push

Status vs. Pending Changes

In Git you can use the command [Git Status]. It shows you all the modified files in your local repository that have not been Pushed. In TFS we have the view Pending changes windows. In the UI it looks quite the same and has the same behavior. Beware of the branches in Git.



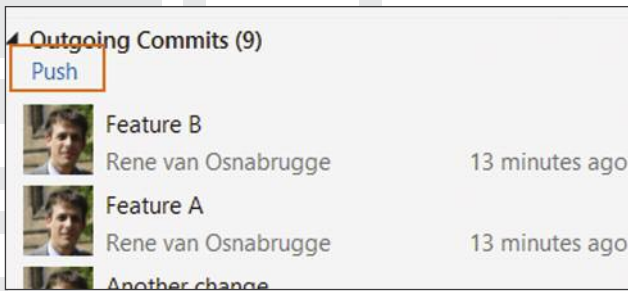
TFS Checkin



View Status in Git

Push vs. Checkin

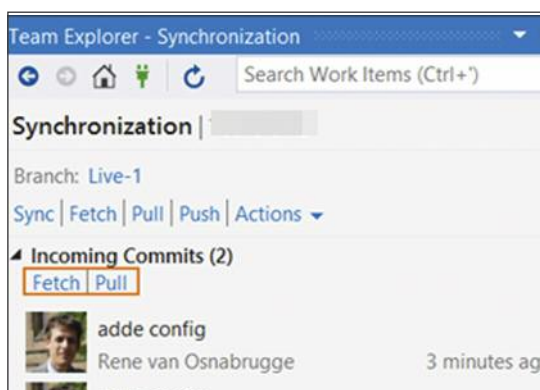
When you want to send your changes to the remote repository where you originated from in Git, you can use the [Push] command. With this command, you send all your local commits to the remote repository. In TFS you use checkin for this. Because there is only 1 central server you always checkin on this server.



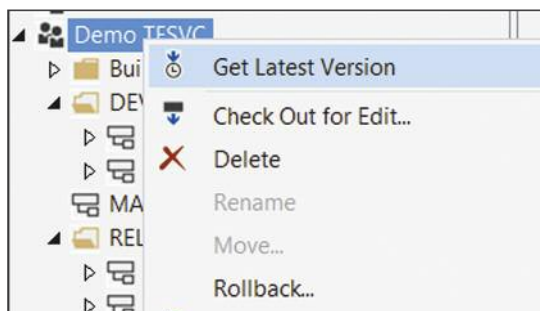
Push in Git

Pull vs. Get Latest Version

To get all commits from the originator repository that were made by others, you can use the pull command. With this command you retrieve all the changes. In TFS we use the [Get Latest Version] of [Get Specific Version] command to synchronize the workspace.



Pull in Git



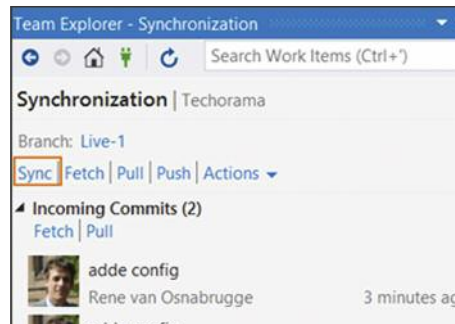
Get Latest Version In TF

Sync vs. CheckIn and Get Latest Version

Sync is not really a Git Command. You use Push and Pull. Visual Studio created a nice graphical button to do this together.

Summary

It is hard to compare 2 version control systems that are conceptually different. But in essence it are both version control systems that you can use to store your changes safely.



Sync button in Visual Studio

Both TFVC and Git have their advantages and disadvantages and it is hard to make a good choice. Git is fast and flexible and most of all supported on all platforms. As a downside it has a steep learning curve. TFVC is fast, robust, enterprise ready and easy to use. With the pointers in this article, try out Git. Use it in your daily scenario and make the choice for yourself. After all it is a matter of taste after all.

Resources:

- Git Basics – <http://git-scm.com/book>
- Announcement of Git and TFS
<http://blogs.msdn.com/b/bharry/archive/2012/08/13/announcing-git-integration-with-tfs.aspx>
- Wikipedia Definition –
[http://en.wikipedia.org/wiki/Git_\(software\)](http://en.wikipedia.org/wiki/Git_(software))
- TechEd 2013 video of Martin Woodward –
<http://channel9.msdn.com/Events/TechEd/NorthAmerica/2013/DEV-B330#fbid=Lb7e4eYEDpC>
- Build 2015 video of Martin Woodward –
<http://channel9.msdn.com/events/Build/2015/3-746>
- Use Git in Visual Studio –<http://blogs.msdn.com/b/visualstudioalm/archive/2013/02/06/set-up-connect-and-publish-using-visual-studio-with-git.aspx>
- A successful Git branching model – <http://nvie.com/posts/a-successful-git-branching-model/>



Rene van Osnabrugge

Lead Consultant

<https://xpirit.com/specialists/rene-van-osnabrugge>



First programming experience with Microsoft HoloLens

Back in January of 2015 Microsoft unveiled the HoloLens, which it calls the first holographic computer running Windows 10. The computer fitted inside a wearable device will add holograms to the real physical world. Holograms are objects made entirely of light, can be 2-dimensional or 3-dimensional and can be viewed from every side. They do not actually exist, but appear to be blended into the real world. Unlike actual objects they have no mass and cannot be touched. Even so, the experience that the HoloLens creates provides a new view of the world.



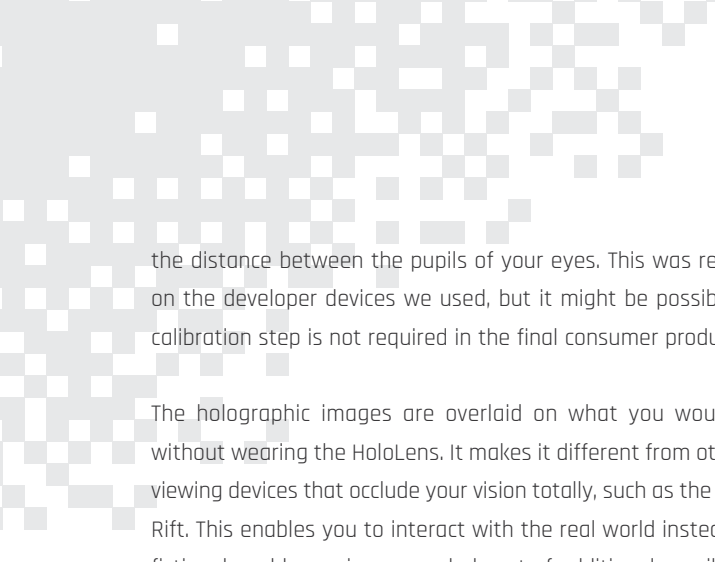
During the //build/ conference in April 2015 more details and applications of the HoloLens were unveiled. Also, the Windows Holographic Platform was announced as the higher level initiative to integrate holograms into the Windows platform, where the HoloLens is a hardware device that uses these capabilities.

A select group of developers were allowed to experience the HoloLens first hand and create a working application for the device. Marcel de Vries and Alex Thissen took the deep dive and worked their way through the Microsoft Holographic Academy guided by Microsoft mentor Emily.



New capabilities offer new possibilities

The HoloLens is an untethered wearable computer that fits on your head with a band. It is like wearing a baseball cap with a large set of sunglasses attached to it. Its band is secured with a knob on the back of the band to make it an exact fit for your head. You can extract and raise the visor to have it fit correctly on your head regardless of whether you are wearing glasses or not. The visor contains multiple sensors, a set of high-definition lenses to display the holograms and a Holographic Processing Unit (HPU) to perform data acquisition and processing of the sensor data. Before you use the HoloLens you need to calibrate it so it knows



the distance between the pupils of your eyes. This was required on the developer devices we used, but it might be possible this calibration step is not required in the final consumer product.

The holographic images are overlaid on what you would see without wearing the HoloLens. It makes it different from other 3D viewing devices that occlude your vision totally, such as the Oculus Rift. This enables you to interact with the real world instead of a fictional world, opening up a whole set of additional possibilities. The HoloLens not only offers visual enhancements, but also positional stereo sound by two speakers on either side of the device.

The HoloLens is a device that has a vast array of sensors and devices fitted inside the headset. The exact type of sensors has not been disclosed yet, but they allow the HoloLens to support the following:

■ Voice recognition

You can issue voice commands that are picked up by the microphones inside the HoloLens. The commands are recognized and translated to string-based commands. These commands work amazingly accurate and direct. You do not have to raise your voice, as the device picks up normal spoken words and sentences.

■ Spatial mapping

The HoloLens can scan the environment (e.g. the room in which you are standing) and map out the objects and surfaces that it contains. Normally you would not see the mapping of the room. It is possible though, to show the mapping with a triangular mesh, so you can actually see what has been scanned. The scan of the environment is cached in some form, so you can quickly turn around and see the correct mapping. A rescan is performed periodically to update any changes, and also to remove any glitches that might have occurred.

■ Gesture recognition

The HoloLens is able to see your hands in the field of view. It can recognize (at least) a closed fist and an index finger. Moving your stretched index finger down and up is a “tap”-gesture, which is mostly used to indicate and initiate an action that would normally be done with a mouse-click. You can compare this to the capabilities the Microsoft Kinect introduced a couple

of years ago.

■ Gaze tracking

Your gaze is comparable with a laser beam coming from the front of the HoloLens. It is centered in your field of view. By turning your head and body you can look around the entire environment redirecting your gaze by doing so. The center point of your view acts as the focus point for cursors projected into the environment. By moving your head and looking towards objects you gaze towards certain objects that can be detected by the HoloLens. This enables you to select those objects and interact with them using gestures or voice commands.

■ Ambient and positional sound

The sound speakers in the HoloLens can produce both ambient sounds, like background noises or music, and positional sound that appears to be coming from a specific source at a certain position in the environment. It might seem similar to normal stereo sound, but the big difference is that you can turn around and face any direction. This means that the sound from each speaker is adjusted for the direction you are facing.

The capabilities mentioned can be leveraged by HoloLens applications to produce unprecedented experiences and possibilities for interactions.

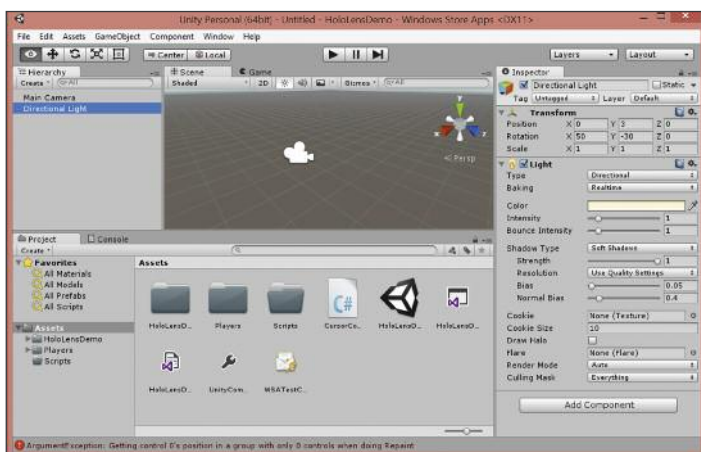
Combining Unity and Visual Studio

You can program applications for the HoloLens in a variety of ways. The applications can be any standard Universal application for Windows 10. They are 2-dimensional windowed applications that you place somewhere in the environment on a flat surface like a wall or a table surface.

More immersive applications use the capabilities of the HoloLens to enhance beyond mere projection into the environment. Such applications use a low level C++ programming model using DirectX, or a higher level managed language (such as C#) combined with the .NET framework.

During the Holographic Academy we developed a Origami demo, where a playground of paper origami objects was placed into the classroom. The playground consisted of a notepad of paper sheets, two folded paper planes and two balls. We added functionality and logic to drop the balls onto the planes and let them roll down

into our real world. These actions are initiated both by voice and gesture commands. Once dropped the balls actually bounced against holographic, but also real world objects recognized by the HoloLens from the spatial map information. In the final exercise we changed the logic such that when the ball dropped to the ground, it show an explosion on the point of impact. After the explosion a virtual hole in the ground was shown. Through this hole there appeared to be a complete new 3D world underneath our classroom floor. Watching into the hole let you view that world from all angles as if it was really there.



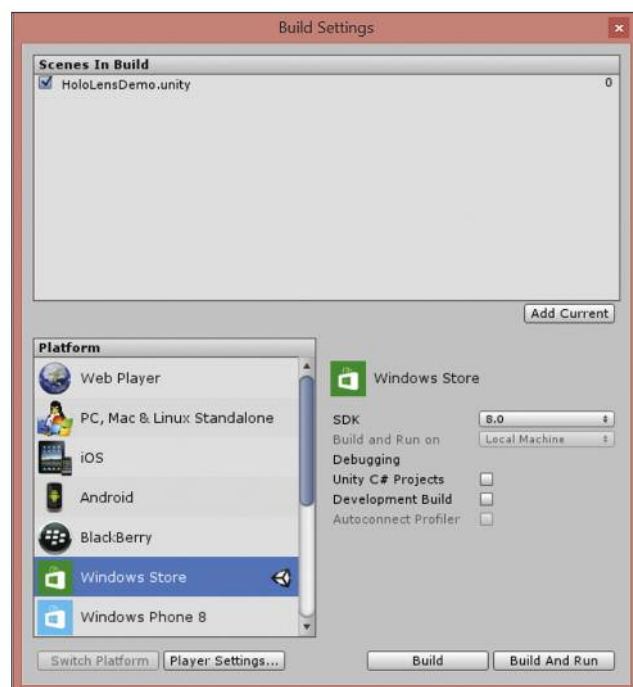
We used Unity 5.0 with some changes and extensions for the HoloLens device. Unity is mainly used for 3D applications and games. It simplifies the use of 3D objects, grids and meshes and has a physics engine to mimic real world interactions between objects and the world. Together with the extensions Unity is a perfect fit as it has all the ingredients to mix real world characteristics and the input and output of the HoloLens sensors and lenses.

Typically you create a scene with a camera in the Unity development environment. The camera views the scene consisting of several objects and the controls allow you to interact with the object and the camera's position and direction. HoloLens brings the combination of the environment and the scene, plus the interaction through the sensors instead of keyboard and mouse. The addition of spatial mapping allows the scene to interact with the actual layout of the room, because it will give the physics engine the contours and dimensions of the outline of the real world objects. Add some script logic to the Unity mix and you can create unprecedented applications.

Programming for the Microsoft HoloLens

When building a HoloLens application the first thing to do is to remove the default camera in the hierarchy of objects. You are viewing the world from the HoloLens and not from the traditional camera. The HoloLens SDK provides a special HoloLens "camera" that you can use in the Unity environment. This is a special stereoscopic camera that is positioned and oriented by yourself and you move around in the environment, turn and look up and down. The HoloLens SDK also provides several entities to be used in the Unity programming environment. For example, it has a Spatial-Mapping object that is added to the hierarchy of objects in the application. It will automatically add the spatial layout obtained by the sensors and the HPU to the scene. The Unity rendering and physics engine take care of the processing.

Unity uses scripts to add behavior and functionality to the entities in the application you are building. The script can be written in either C# or JavaScript. We created the scripts in C#, by simply clicking the Scripts pane from Unity and adding the appropriate script. You couple these scripts to the object or artefact in your Unity scene, thereby attaching the functionality in the script to the entity.



A simple double-click on a script transitions from the Unity environment to Visual Studio. The Visual Studio solution you created earlier contains all the scripts you have added.

It also holds any other C# code that you need to use, such as helper classes or logic.

In general code and logic inside Unity is related to entities. The key events for entities is the startup and an update of the entity's state for rendering another display frame.

Let's look at an example scenario of creating a behavior: cursor placement. A HoloLens application can use a cursor to indicate the direction of your gaze and to provide visual feedback of the actual object or surface you are interacting with. You need a visual mesh to render as the cursor and some logic to define the behavior. The visual mesh here is a donut shape that we can then place on the place where we look at, so it looks like we have a visual cursor following our gaze.

The code fragment below shows the skeleton of such a class with the particular startup and update logic. The implementation is indicative of the abstraction level of the SDK.

```
public class WorldCursor : MonoBehaviour {
    private MeshRenderer meshRenderer;

    // Use this for initialization
    void Start()
    {
        // Grab the mesh renderer that's on the
        // same object as this script.
        meshRenderer = this.gameObject.GetComponent<MeshRenderer>();
    }

    // Update is called once per frame
    void Update()
    {
        // Do a raycast into the world based on
        // the user's
        // head position and orientation.
        var head = StereoCamera.Instance.Head;
        RaycastHit hitInfo;
        if (Physics.Raycast(head.position,
            head.forward, out hitInfo))
        {
            // If the raycast hit a hologram...

            // Display the cursor mesh.
        }
    }
}
```

```
        meshRenderer.enabled = true;
        // Move the cursor to the point
        // where the raycast hit.
        this.transform.position =
            hitInfo.point;
        // Rotate the cursor to hug the sur-
        // face of the hologram.
        this.transform.rotation =
            Quaternion.FromToRotation(Vec-
            tor3.up, hitInfo.normal);
    }
    else
    {
        // If the raycast did not hit a ho-
        // logram, hide the cursor mesh.
        meshRenderer.enabled = false;
    }
}
```

The Start method typically does one-time initialization. In the code example it will acquire the mesh of the donut we renderer for the cursor. Inside the Update method you perform the following steps:

1. Acquire the direction of the gaze as a vector in 3D space from the HoloLens.
2. Perform a raycast in the direction of the gaze and test whether an actual object or surface was hit. Casting a ray is like shooting a laser beam straight from the origin, in this case front of the HoloLens. The hit test of the beam tells whether have focused on something or are just staring into an empty space. This raycast is available in Unity and not HoloLens specific.
3. If there was a hit, determine the normal vector of the hit location. The normal is the vector perpendicular to the surface (of the object).
4. Perform a transformation (rotation) of the cursor to match the direction of the normal.
This will make the cursor follow the curves of the object or surface, giving it a natural feel of clinging to the real world objects or holograms. This is done by a standard 3D operation called a Quaternion rotation.
5. Display the cursor by rendering its mesh.

By adding the cursor mesh and attaching the logic from the World-Cursor class we get the cursor behavior in just a couple of lines of code.

Once your application is built from both Unity and Visual Studio, you can deploy it to the HoloLens device through the USB cable that connects your development computer and the HoloLens. Running the application is as simple as selecting Run without Debugging from the Debug menu in Visual Studio. The HoloLens provides a web interface that you can use to upload programs and start and stop them. You can just open a browser window and navigate to the HoloLens' IP address, which will show you a web page to interact with. On this page you can upload the standard windows 10 Universal App packages. Once uploaded they can be run from either this web interface or straight from Visual Studio.

In conclusion

Programming and testing our first HoloLens application was an incredible experience. We had no prior knowledge of Unity, but plenty of years experience programming in C# and a variety of frameworks. The use of Unity takes some getting used to. Fortunately there is an abundance of online tutorials and documentation and an active community. With the help of the mentor and the documentation we were able to create our

origami demo within 4 hours and have it running on the actual HoloLens devices. This was a truly amazing and inspiring experience.

In the Windows 10 timeframe Microsoft will release the HoloLens device and make it available to the general public. Presumably before that time developers might get their hands on the device and SDK to start developing compelling and innovative applications for the HoloLens. There is an incredible amount of new possibilities available applications for the HoloLens. We can't wait to get started for real and put holograms to use. The question remains what the product release date and initial costs will be. Many thanks to Emily and the HoloLens product team for assisting at the Holographic Academy and reviewing this article.



Alex Thissen

Lead Consultant

<https://xpirit.com/specialists/alex-thissen>



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.

Cross Platform Mobile Development Jumpstart with Xamarin

https://training.xebia.com/mobile/cross-platform-mobile-development-jumpstart-with-xamarin?utm_source=xpirit&utm_medium=website&utm_content=title-link&utm_campaign=xpirit

Architecting hyper scale solutions for the Microsoft cloud

https://training.xebia.com/architecture/architecting-hyper-scale-solutions-for-the-microsoft-cloud?utm_source=xpirit&utm_medium=website&utm_content=title-link&utm_campaign=xpirit

Enterprise Development with NServiceBus

https://training.xebia.com/developer-skills/enterprise-development-with-nservicebus?utm_source=xpirit&utm_medium=website&utm_content=title-link&utm_campaign=xpirit

Professional Scrum Developer (PSD) with .NET

https://training.xebia.com/scrum/professional-scrum-developer-psd-with-net?utm_source=xpirit&utm_medium=website&utm_content=title-link&utm_campaign=xpirit

Lessons learned: migrating an N-Tier web app to microservices

Microservice architectures are all the craze nowadays. In my opinion it is the next evolutionary step for loosely coupled and scalable architectures. It builds upon the foundations we laid with Service Oriented Architectures, when it was not spoiled by Erroneous Spaghetti Buses or people who thought it was a good idea to throw a bunch of SOAP webservices at a problem and call it SOA. After all, it's service orientation, right?

One important thing is that you learn to disassociate webservices with the concept of services. They are not the same, and mixing these two has led to the abominations we have started to link to "SOA". In SOA, a service is the technical authority for a given business domain, and can use multiple technologies to implement its goals.

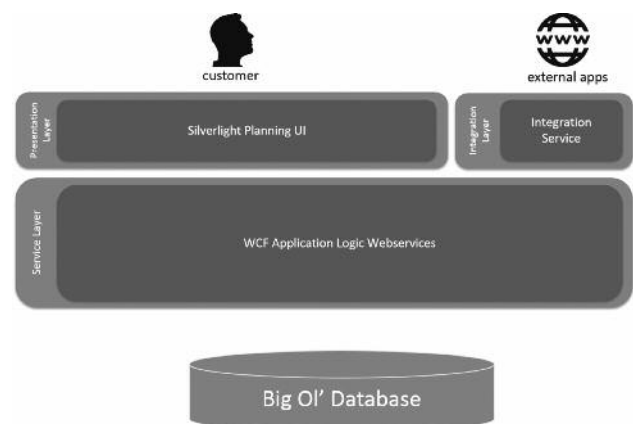
In this magazine, you can read more about what Microservice architecture is according to Xpirit, so I will not go deep into its definition or characteristics. Rather, I want to focus on how to get there if you do not have a fresh green field to build your software in. What if you have a system that consists of said SOAP webservices, an N-tier web application with a fair amount of legacy? How do you transition that to microservices? And why should you, even?

This article is based on my own lessons learned while migrating such an application to a more loosely coupled architecture, using the NServiceBus framework as its foundation. But even if you are not into NServiceBus, the concepts and steps taken are basically technology agnostic. NServiceBus just makes things a lot easier. You could also look at using MassTransit, RabbitMQ or Azure Service Bus for messaging. The key requirement is that you have some infrastructure to facilitate in hosting your autonomous services and to provide the transport for your messages and events.

So what is wrong with N-Tier anyway?

The application I inherited when I came into the project was a multi-tenant SaaS application that dealt with scheduling work for employees in supermarkets, a Workforce Management System.

It consisted of a single database per tenant and a Silverlight front end backed by WCF SOAP webservices. While Silverlight may seem like an obsolete technology now, this is not really relevant because we would have had the same issues if the UI was built as a Single Page Application with AngularJS.



We had a couple of challenges:

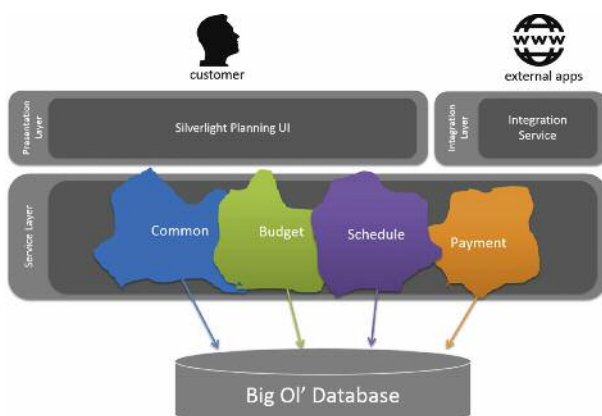
- We were about to scale out to many more customers, and subsequently: users, and we knew the system wasn't able to handle that at the time, for reasons that will become clear in this article
- We also had some work to do to make the system a real multi-tenant system
- There was an increasing demand for standardized integration with other external systems, such as HRM or Payroll systems and the existing solution was based on point-to-point webservices and under high load, was known to lose data.

So what is wrong with this multi-tier architecture? We learned that layering is good because it helps separation of concerns and abstracts many technicalities as you move up the stack. Above all, layers provide loose coupling, right?

Let us take a look at this conceptually first: from 10,000 ft., this architecture looks like basically any web based application. It is a pretty generic structure. And there is the first problem: the architecture is driven by a technical breakdown. We have our database, our service layer containing application logic, and our UI layer. But what does this architecture say about what the application actually does? Sure, if we zoom in a bit, there are areas in the code to be found, such as Budgeting, Scheduling and Payment, but this is not its top level structure.

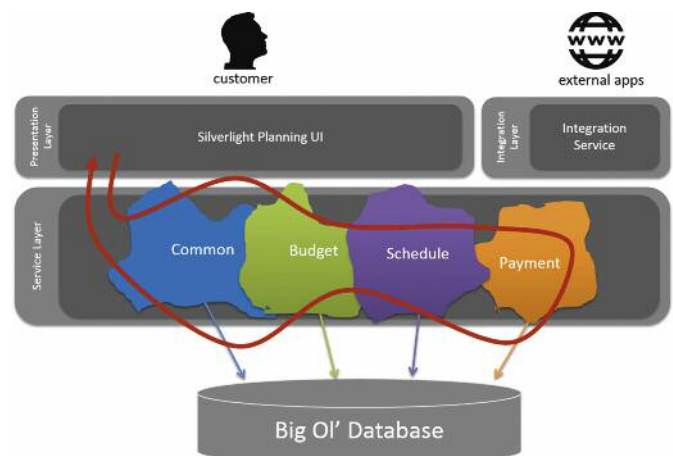
Moreover, this architecture forces us to shoehorn all functionality into these layers, forcing us to implement repositories, catalogs, webservice facades, webservice client wrappers, UI modules and end user screens for every feature or entity what was added. Layers upon layers upon layers, with the occasional data mapping in between.

Loose coupling you say? Try changing the datatype of a field, or adding a field to an entity and you will see that it is not loosely coupled at all. The point is: apparently there is inherent coupling from top to bottom within a certain functional area of the system and no amount of layering is going to decouple this. Adding layers will just amount to more code, more plumbing, more indirections and needless abstractions. Why bother? This vertical kind of coupling is quite natural and you should not fight it. What is even worse: this technical layered approach encourages developers to create high and unwanted coupling within a layer. On our case, the architecture of the system on a 300 ft. level, looked like this:



Why are these four functional areas irregular shapes, and why do they appear to overlap? It is because none of these modules had a clear boundary, and all modules took the freedom to peek and poke in each other's database tables, or use each other's repositories to achieve their goal. So in essence, we were dealing with a monolith.

For some of these webservice operations, the thread of execution looked like this:



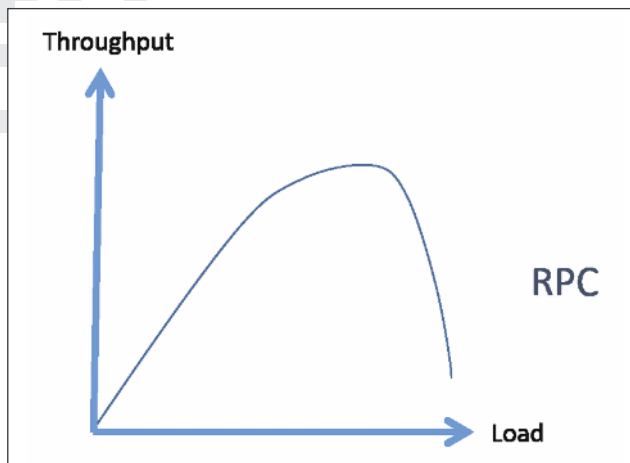
It went all over the place, basically because the way the system was modeled led to the mindset that we had to do everything we needed to do to make the system consistent within that same webservice request, before returning to the calling front end application. The upside of this is that the system is consistent across all modules after the operation completes. Immediate consistency.

But the downsides are much bigger:

- Hard coupling between the functional areas of the system, resulting in nearly unmaintainable code;
- Temporal coupling, because the caller has to wait for every request to complete
- A huge scalability problem because the more the system has to do, the longer the operation will take, thereby blocking precious IIS threads in the resource pool, and finally;
- Locking and blocking on the database because everything is handled inside a single huge transaction.

Under the increasing load we were adding to the system, we saw it break with exponentially growing execution times, database timeouts or even HTTP timeouts because the Silverlight client

would give up waiting for an answer. This is what tends to happen with these RPC style webservice interactions. As load increases, throughput stagnates and breaks down at some point:



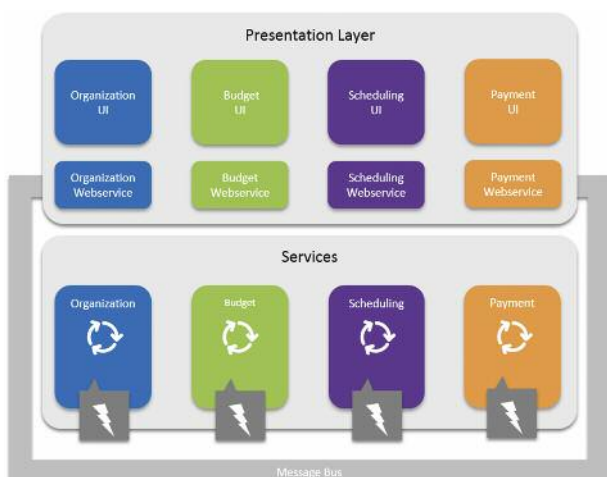
If it's not by layers, then how do we decouple?

This is where the principles from microservice architecture come in... Like Martin Fowler describes, a microservice is:

[...] built around business capabilities and independently deployable by fully automated deployment machinery.

Roughly translated: a microservice is bounded by its (business) domain, and decoupled in nature because it operates autonomously. (SOA, anyone?)

If we look at the system from this angle, surely we should be able to transform the irregular shapes in the earlier figures to more clearly defined ones. Something like this:



That looks much more structured. We see that some of the layering is still there: we had that Silverlight front end that runs on the end user's computer and it needs to cross the network in order to reach the server logic. HTTP and webservices are fine for that. Note that these webservices are now drawn inside the "Presentation Layer" box. The reason for this is that I consider them private webservices, used solely to serve the UI. Their logic is part of presentation, and the physical webservice layer is only there to cross that network boundary. This way, from an architectural point of view layering only becomes a technicality.

We can clearly see that each microservice spans from top (UI) to bottom (service logic and data storage). As mentioned earlier, vertical coupling is natural. But horizontally, there is no direct coupling anymore, which is the type of decoupling we want to achieve. Communication between (micro)services happens through events that are published by microservices, and I have introduced a Message Bus to facilitate that communication. In our case, it was NServiceBus that facilitated the publishing and receiving of these events.

"But, where is the database?" you may ask. Well, in the true microservices mindset, each microservice has its own datastore, which can be anything ranging from an RDBMS to a Document Database to a file share. It is part of the service.

Breaking up our application

So, how do we transition our existing codebase from A to B? Here is a five step plan to help you out:

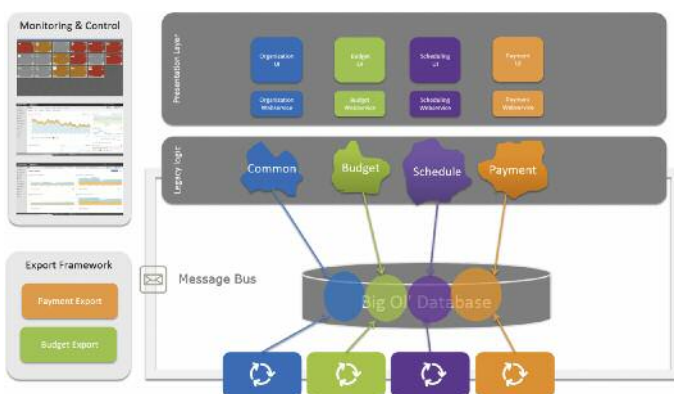
1. Together with your domain experts, identify significant business events in your system. E.g. "employee created", "employment changed", "budget finalized", "payment finalized". Start publishing these events from your application, and roll out more and more events with every update of your system, even if you don't handle them yet.
2. For new features, preferably the ones that happen "at the edge" of your system, start using these events as triggers to do the required actions. Examples are: export functionality, calculating reports in the background, generating tasks for a task list, deduplication of data for performance reasons. This makes it easy to add new modules to the system without touching much of the existing code.

3. Start thinking about how to create a composite UI to tie these services together. As you break up your system into more autonomous components, your UI must remain coherent to the end user.

4. Over time, migrate existing code from your webservices to the autonomous services where possible. Shrink the webservices down to the bare essentials: make sure the webservice does everything it needs to do to get a (command) message on the message bus, and return as soon as possible. This will increase the responsiveness of your webservices, and free up IIS request threads to handle more load. Of course, if the front end requires data, e.g. with a query, you should not shoehorn asynchronous messaging into the scenario. It is OK to call a repository directly to retrieve the necessary data.

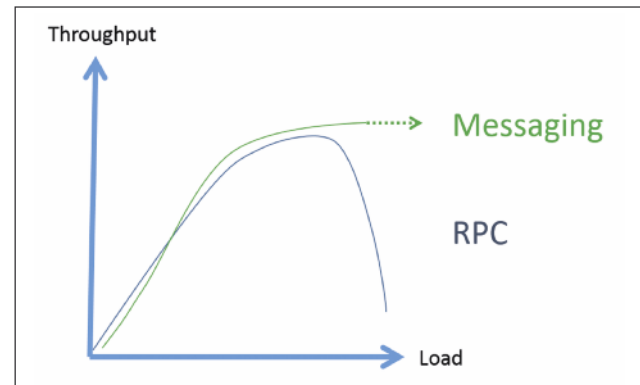
5. When you are ready for it, split up the database so that you can give each service its own storage.

Your system will probably look similar to this during the transition:



The irregularly shaped modules will start to shrink and lose their overlap. Some modules may even get a different name because you have found their purpose. In our case, a chunk of code with the generic name "Common" was renamed "Organization" because it dealt with concerns surrounding the customer's organizational structure. Along the edges of the system, connected via the Message Bus, new services will appear that leverage events to do their work. And along the way, inside your database, it will become apparent which tables belong to which microservice.

Adding messaging to your system will generally give your system better throughput. Up to a certain point the throughput will keep up with the load on the system, but at its peak it will remain constant because we introduced decoupling:



Side effects

With this transition to microservices, you might see some side effects. Some of them are a result of a different style of thinking once you let go of the "everything needs to happen inside one web request" mindset.

Task based UI paradigm

As you introduce messaging and events, you will start to think in terms of commands to fulfil some user intent and events that arise from those actions, in turn triggering (sub) processes in other microservices in the background. A Task Based UI is a natural fit for this command-style interactions. For example: once a budget is finalized, a task will come up for the user to start working on a schedule. But also: actions from the UI will become much more intentional. Instead of CRUD-like data interaction, where the user edits a set of data fields and presses "Save", buttons and actions will be much more modeled to capture the actual user intent: "renew employee contract", or "finalize budget".

Keep in mind that a task based UI requires a different mindset from everyone in the team. Your domain experts and system analysts (those who write your user stories or system specs), have to be the first to understand and embrace this style of thinking. The same goes for eventual consistency. You must engage with them in a discussion about where (ACID) transaction boundaries lie, and where you can divide things up. It all has to make sense from a functional angle as well.

Performance

Do not think that throwing in a message bus and putting commands on a queue alone will alleviate your performance problems. If the code that handles these commands is still the same spaghetti code that does too much, this execution will still take too long. While the system will be able to handle this load more comfortably, the end-to-end waiting time for the user may become much longer, because of the queueing. Only migrate code from your webservices if you can successfully break it down into distinct and (ideally) small responsibilities. This usually involves rewrites or large refactorings, so this is something that will take time and multiple releases to get right.

Closing thoughts

Because you are in a brown field situation, prepare to make concessions at first. You will have to take it one step at a time, and sometimes it makes sense to release the system with some code duplication while you move logic to the background in increments. But always be aware of when you are creating this type of technical debt: make sure to clean it up at the first opportunity.

What about my Big Ol' Database?

It is unrealistic to think that you can just give each newly defined microservice its own database right from the start. You will find that there is a lot of hidden coupling inside or via the database which you need to disentangle step by step. Therefore it is more realistic to start out by keeping that same single database, and instead try to sketch the microservice boundaries into the data-model as you assign sets of tables to each microservice's area of responsibility. Over time, you will find out where you can cut out parts of the database and move it to a separate store, and where you will need to apply data duplication by means of events across services. Just as long as you manage to designate which microservice is responsible for that part of the data, i.e. which service is responsible for the creation of that data, and who only consumes it?

Monitor, measure and learn!

Add monitoring and instrumentation to your system as soon as possible. Especially as you start publishing and handling events, you have to have insights into whether all these autonomous services are doing their job, let alone how the overall system performs. We used New Relic to handle that at the time but there

are also other options, such as Application Insights, which is part of Microsoft's Azure offering.

Once you have monitoring in place, it can even drive your transition:

■ Usage analysis

Find out from your usage statistics which parts of the system are used the most, and focus on transitioning those parts of the system first. You may even find out that some parts are never used, and you can decide to remove them over time, or find other ways to achieve the same goal.

■ Performance analysis

Look for the performance hotspots in your system. Which web-services are causing the most contention or slowness in the system? Try to break those up first.

The most important thing is to learn from your system as you perform the transition to microservices. I found things I had never anticipated just by studying our user's behavior.

Be prepared to take around a year or more for a transition like this.



Roy Cornelissen

Lead Consultant

<https://xpirit.com/specialists/roy-cornelissen>

Start with Visual Studio Release Management vNext

Team Foundation Server 2013 Update 3 came with Visual Studio Release Management vNext. vNext is, next to the deployments with agents, another way of doing deployments with VSRM. Deployments to machines, without having to install an agent, is the most important feature of vNext.

This is because VSRM vNext uses PowerShell Desired State Configuration (PowerShell DSC) as the tool/engine to execute PowerShell on different machines.

Although TFS 2015 will contain a whole new Release Management implementation, this article guides you through a setup of the current version of Release Management

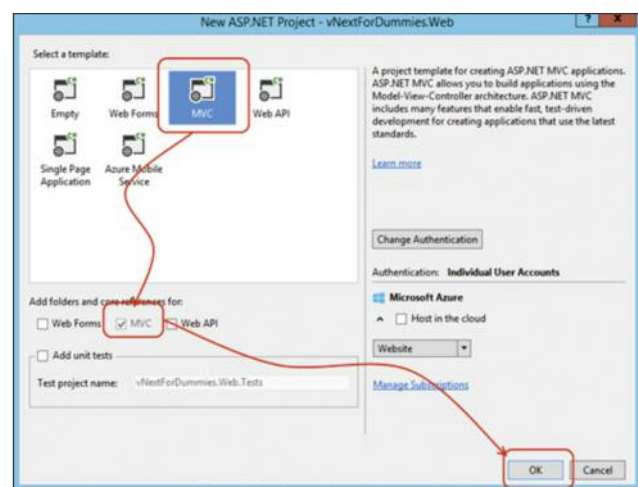
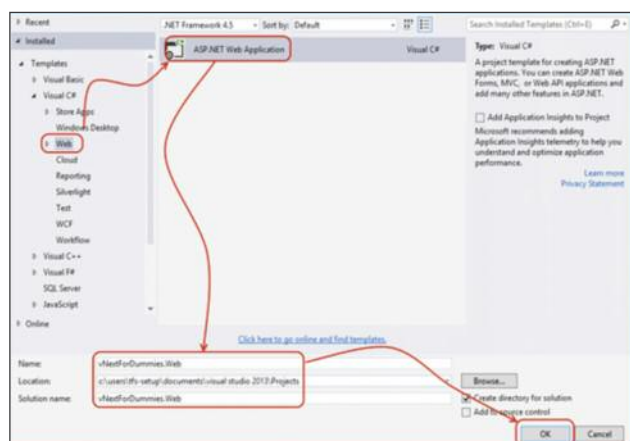
1. Check your prerequisites

In order to use RM vNext you will need the following prerequisites

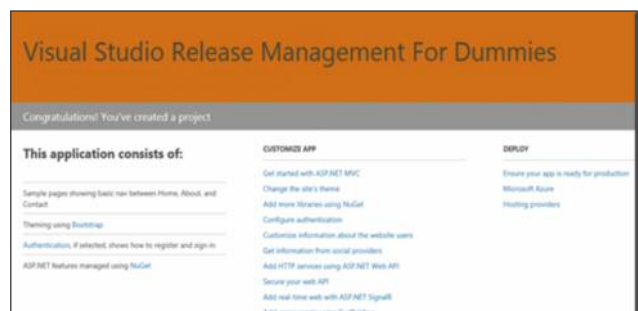
- Azure subscription where I can create a Virtual Machine
- Team Foundation Server 2013.4 or 2015 RC installed
- Release Management Server + Release Management Client 2013.4 or 2015 RC installed
- TFS Build Server + Agent 2013.4 or 2015 RC installed
- Visual Studio 2013.4 (Pro/Premium or Ultimate) or 2015 RC
- An empty Team Project (RMvNextForDummies)

2. Create the simplest web application ever

- Start up your Visual Studio and create a new Web Application

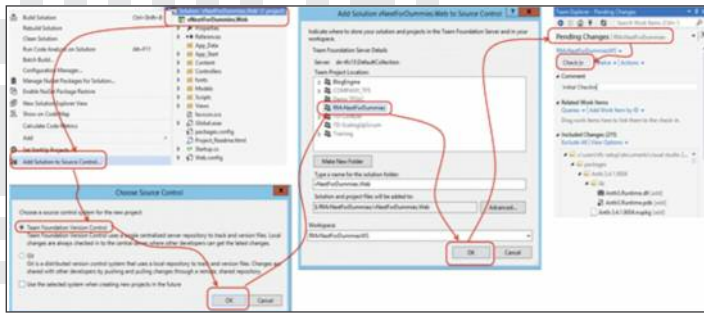


- Run to test the solution. If you want to make it pretty you can always change the color or the title.

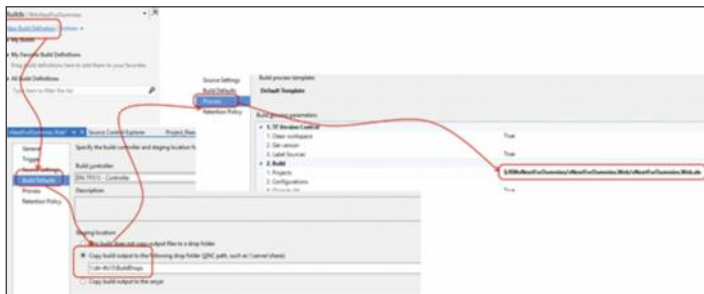


3. Check in Solution and create a Build

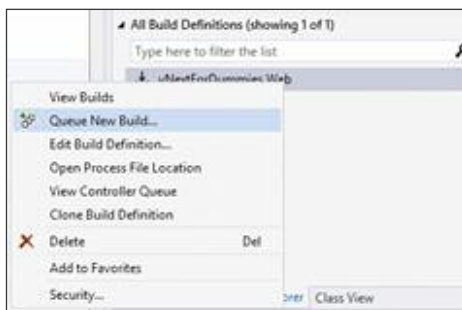
Check in the solution in Source Control in your newly created Team Project. Easiest way is to just right click the solution and choose [Add Solution to Source Control], choose the right Team Project to add the solution to, Navigate to the [Pending changes] hub in Team Explorer and check in your changes.



Navigate to the [Build Hub] in Team Explorer and create a [New Build Definition]. Leave everything default. On the [Build Defaults], fill in a valid drop location, and on the [Process] Tab, make sure you selected the solution you just created.



MSDN Documentation can be found here
Run the build to check if it builds successfully!



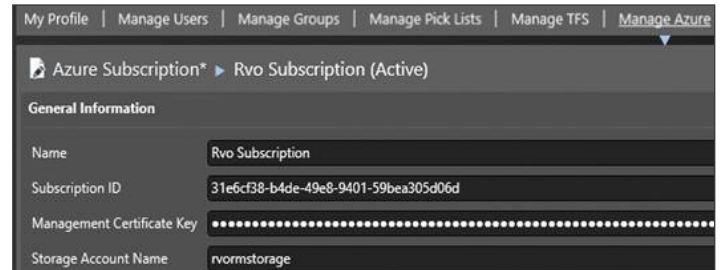
4. Set up Release Management to use your Azure account

Start up the Release Management Client and navigate to the [Administration | Manage Azure] Tab. Click [New] and fill in the details that are asked. The information that should be filled in is described perfectly on MSDN. Click this link and follow the steps described here.

Now create a Virtual Machine with Windows 2012R2 on Azure. Do nothing yet on this virtual!

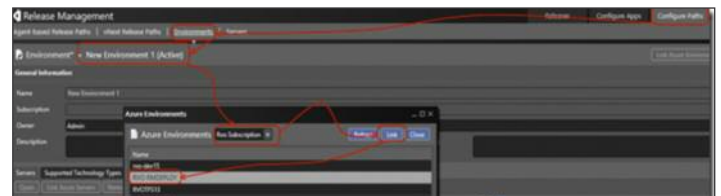
<http://www.visualstudio.com/get-started/deploy-no-agents-vs#SetupAzure>

Get the SubscriptionID and ManagementCertificate from the publishsettingsfile you downloaded. Create a new storage account on Azure especially for using in Release Management. Use the name of the storage account as [Storage Account Name]



MSDN on how to create a storage account

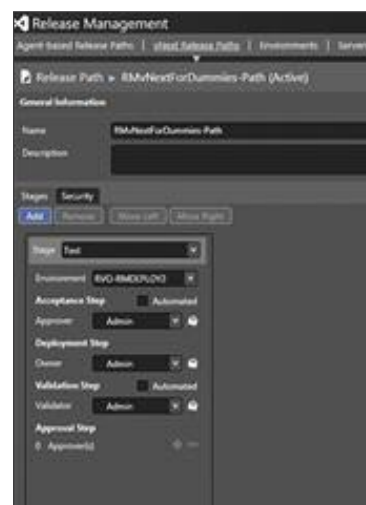
Create a vNext Environment and link your newly created Azure Server

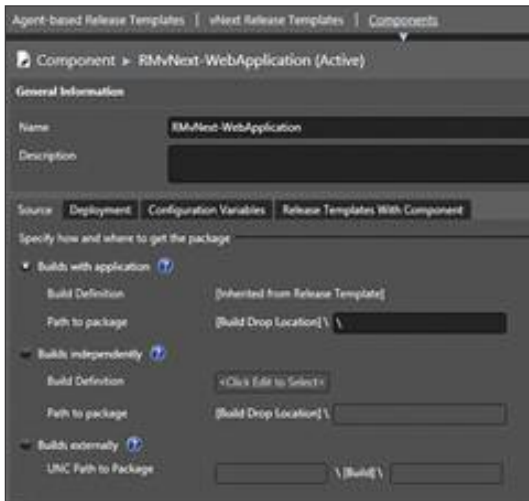


5. Create a Release Template and Path

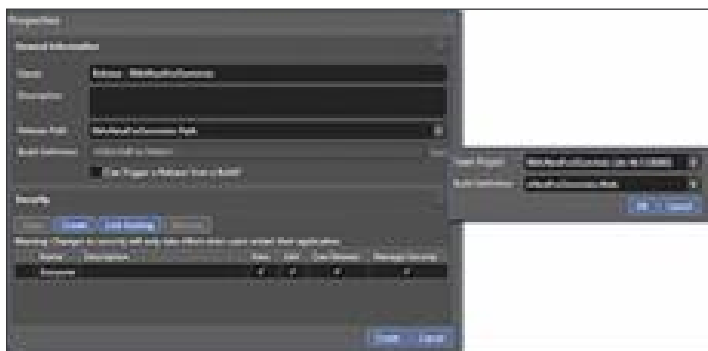
No you need to create stages (our deployment pipeline) and a release template by following the steps described on MSDN (<http://www.visualstudio.com/get-started/deploy-no-agents-vs#CreateReleaseTemplate>). Keep it simple and add only 1 stage!

This is my result for the Release Path





My Component

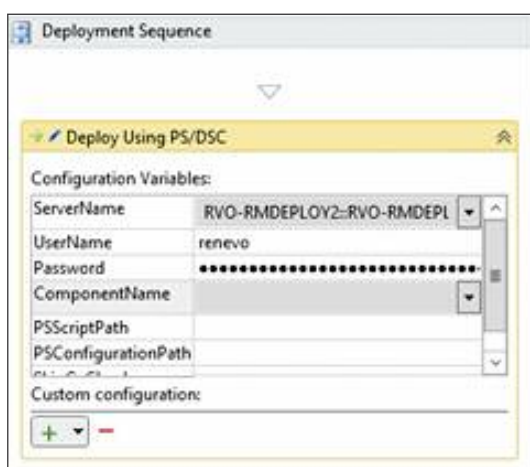


And my Template (still empty)

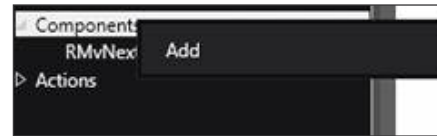
6. Start my first Release

Now. No further hassle, I want to start a release!

Drag the [Deploy using PS/DSC] to the surface.

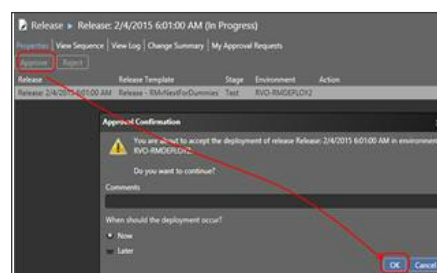
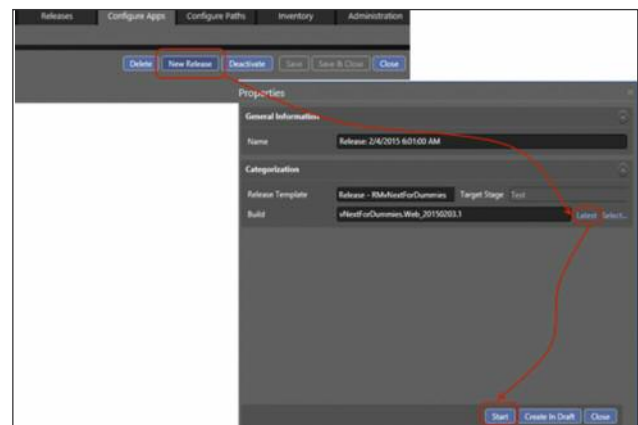


When you try to choose the [Componentname] it cannot be chosen yet. In order to do this, you need to add the component to the toolbox first. Right click the components "folder" in the toolbox and add the component.



Now select the ComponentName and trigger a [New Release]. Select the latest build and start. You'll end up in an approval screen where you can approve the release of the first stage. Approve it and watch what happens.

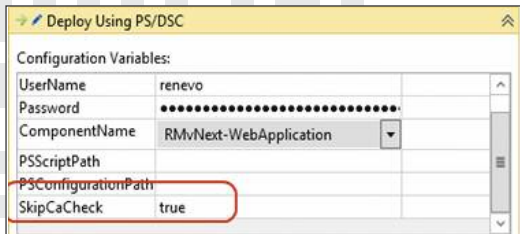
Sure! I know there is no PowerShell DSC script yet, let's try to make the communication work.



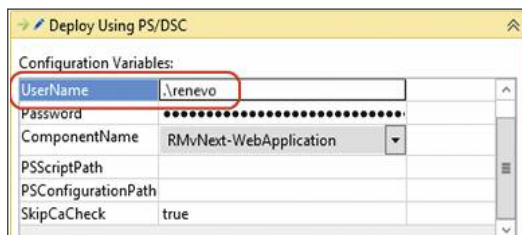
7. Troubleshoot and make it work

When your deployment fails, take a look at the following things:

- Open your firewall ports for PowerShell (default on 5986 and 5985)
- When deploying to a non-domain machine, skip the certificate check in Release Management or upload a certificate from the target machine (as described here: <http://fabriccontroller.net/blog/posts/using-remote-powershell-with-windows-azure-virtual-machines/>)



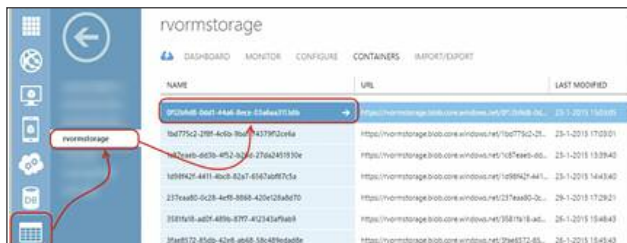
- Make sure you type your username as .username. So with the period



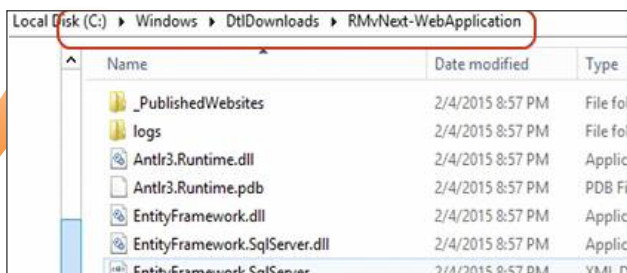
8. Check some artefacts

We have our end-to-end scenario. Deploy (doing nothing yet) succeeded and we are in the validation step. The steps performed are:

- Upload all the build output to your azure storage. You can check this by logging in to your azure account, navigate to storage, select the storage you configured in RM and choose container. There you see the files from the build.



- On the target machine you have a directory [DtdDownloads] in your c:\windows (c:\windows\dtddownloads). Here you find all the files downloaded from storage and ready for further processing on your machine.



9. Create a simple DSC script that we can execute

Now that we have our connectivity, we can start building some DSC. There are some good posts around the internet about DSC and also some in combination with Release Management. You can find those in the resources section at the bottom of this post.

For now, I will suffice by copying the website bits to the inetpub/wwwroot directory on the target machine. The DSC script we want to execute must be send to the server as well. The easiest way to do this is to make the DSC part of the build. Open your web application, and add a folder DSC to your web application. Add a file CopyWebSite.ps1 to this folder and put this in the file.

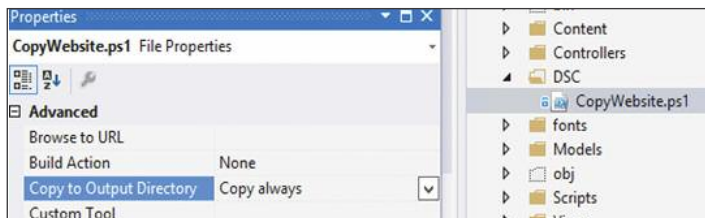
```
configuration FullSetup
{
    node MACHINENAME {
        WindowsFeature IIS
        {
            Ensure      = "Present"
            Name        = "Web-Server"
        }

        WindowsFeature ASPNet45
        {
            Ensure      = "Present"
            Name        = "Web-Asp-Net45"
            DependsOn   = "[WindowsFeature]IIS"
        }

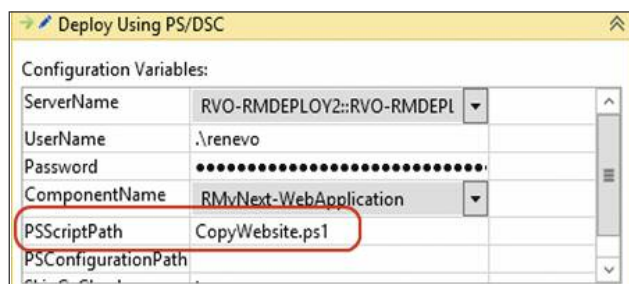
        File CopyDeploymentBits
        {
            Ensure      = "Present"
            Type        = "Directory"
            Recurse      = $true
            SourcePath   = join-path $application-
Path "_PublishedWebsites"
            DestinationPath = "C:\inetpub\wwwroot"
            DependsOn   =
"[WindowsFeature]ASPNet45"
        }
    }
}

FullSetup
```

Make sure you replace MACHINENAME with the name of your target Azure machine. Make sure you set the file properties in VS to Copy Always so that your file ends up in the build.



In your Release Template, add this file to the PSScriptPath



Check in the file, run a build and start a new release. When this succeeds, you have configured an IIS, Framework 4.5 and copied the bits of your build to a directory.

10. Try more with DSC and RMvNext

Now try more with DSC. You can install applications, configure firewalls, set security. All scenarios that are very useful when doing your deployment. Follow the posts listed below for more advanced use.

Summary

Release Management is the bridge between Development and Operations. Using PowerShell DSC and storing your server configuration as code in Source Control allows you to really automate your deployment pipeline. Release Management vNext allows you to do create these pipelines and allows you to enable your approval workflow to production. And realize this with TFS 2015 there is much more to come !

Resources

- Jasper Gilhuis' - Curah on Release Management (a selectin of links)
- <http://blogs.msdn.com/b/visualstudioalm/archive/2014/07/07/how-to-deploy-to-standard-or-azure-environments-in-release-management-2013-with-update-3-rc.aspx>

- <http://www.colinsalmcorner.com/post/using-powershell-dsc-in-release-management-the-hidden-manual>
- http://blogs.msdn.com/b/musings_on_alm_and_software_development_processes/archive/2014/11/21/release-management-and-dsc.aspx
- <http://fabriccontroller.net/blog/posts/using-remote-powershell-with-windows-azure-virtual-machines/>
- <http://www.visualstudio.com/en-us/get-started/deploy-no-agents-vs.aspx>
- <http://blogs.msdn.com/b/visualstudioalm/archive/2014/07/07/how-to-setup-environments-for-agent-less-deployments-in-release-management-release-management-2013-with-update-3-rc.aspx>
- <http://blogs.msdn.com/b/visualstudioalm/archive/2014/07/22/deploying-using-powershell-desired-state-configuration-in-release-management.aspx>
- <http://roadtoalm.com/2014/09/24/silently-install-and-configure-a-tfs-build-server-with-powershell-dsc/>

DSC

- <https://gallery.technet.microsoft.com/scriptcenter/DSC-Resource-Kit-All-c449312d>
- <http://blogs.msdn.com/b/powershell/archive/2014/08/07/introducing-the-azure-powershell-dsc-desired-state-configuration-extension.aspx>
- <http://blogs.technet.com/b/privatecloud/archive/2013/08/30/introducing-powershell-desired-state-configuration-dsc.aspx>
- <http://blogs.msdn.com/b/powershell/archive/2014/04/03/configuring-an-azure-vm-using-powershell-dsc.aspx>
- <https://technet.microsoft.com/en-us/library/dn249921.aspx>
- <http://blogs.technet.com/b/privatecloud/archive/2014/04/25/desired-state-configuration-blog-series-part-1-learning-about-dsc.aspx>
- <http://www.colinsalmcorner.com/post/install-and-configure-sql-server-using-powershell-dsc>



Rene van Osnabrugge

Lead Consultant

<https://xpirit.com/specialists/rene-van-osnabrugge>



API MANAGEMENT

It is for vendors of products very difficult to make mobile applications for every possible mobile platform. Of course by preparing and making a good responsive mobile website available will enhance your reach. Not getting the most out of the mobile devices is the downside of a mobile website.

Because of this vendors make their back-office via API's available to the world. As a vendor you want earn some money on the usage of your API. This means you need a Developer portal where you can divide developers in groups, generate help pages, provide demo/example code, monitoring of the usage, place to get issues, FAQ, but also block or blacklist certain developers/users/applications. In short there is more needed then simple provide some webservises.

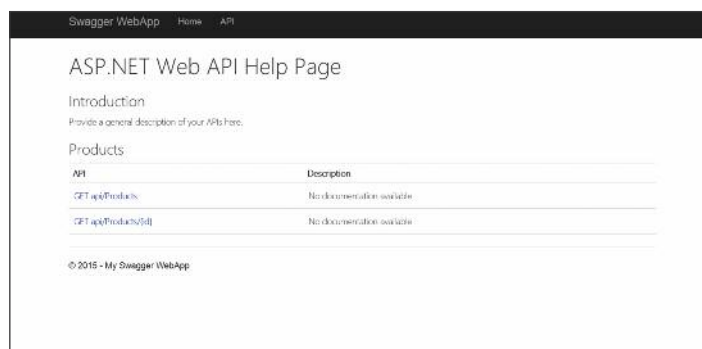
A while ago I had such a dream too. I wanted to create an evaluation app for the SDN. Attendees of a SDN event should be able to fill in an evaluation form via their mobile devices. The evaluation was stored on Azure in a datastore. Of course I did not want to store my tokens/connection strings/passwords etc with my mobile app. And I am not able to create an app for every platform, so I definitely did not want to share my secrets with some unknown/wild developers.



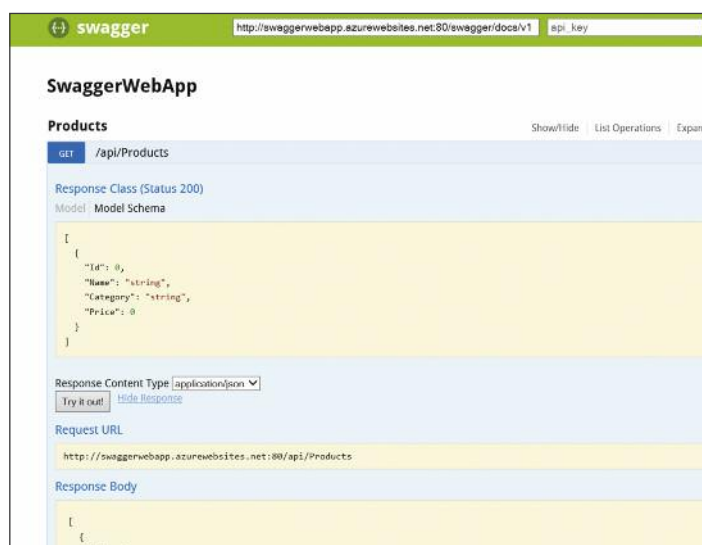
So I thought of preparing some WebApi services and expose them. But like I described above, I needed a portal to explain my services and their responses. That is a lot of work for a small API like this. This is my portal now; <http://sdnevalapp.azurewebsites.net/>.



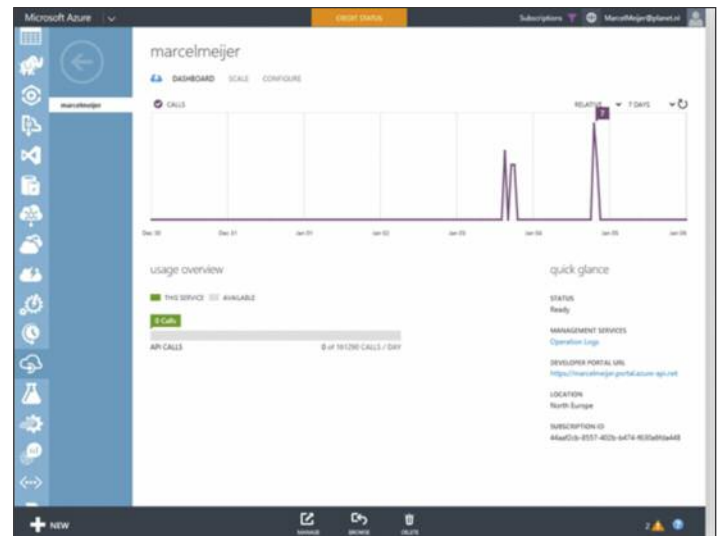
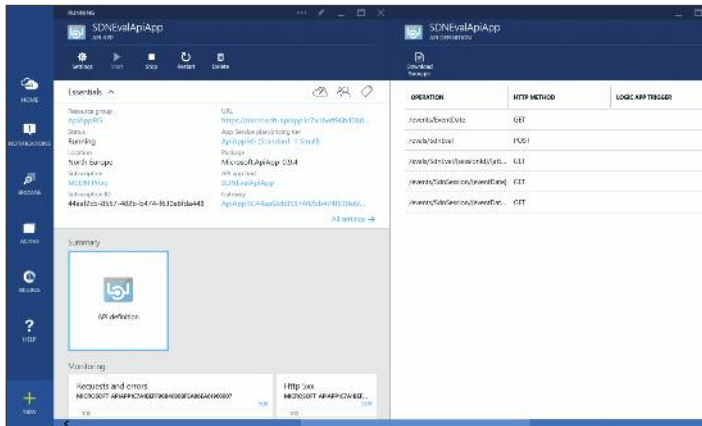
Ok, this solution was an older one. Made in the period where the project template for WebApi wasn't integrated in Visual Studio. If you use the proper WebApi template you will get some sort of documentation by default. Which depends most of your documentation in code.



To make this help page more interactive I could use Swashbuckle (NuGet package). Your users get more details and even can try the methods.



If you are using the newly introduced API Apps you get the swagger stuff out-of-the-box. It is also nicely integrated in the preview Azure portal. At this moment the try-it-out part is not implemented, perhaps it will come soon on the fast changing preview portal.

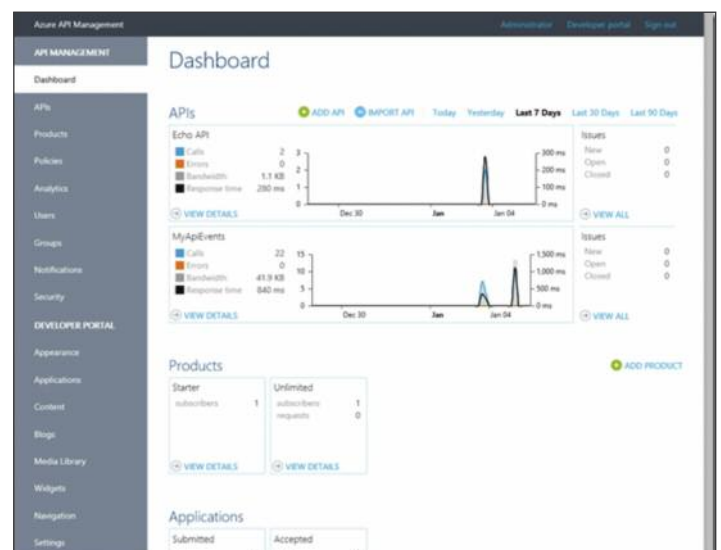


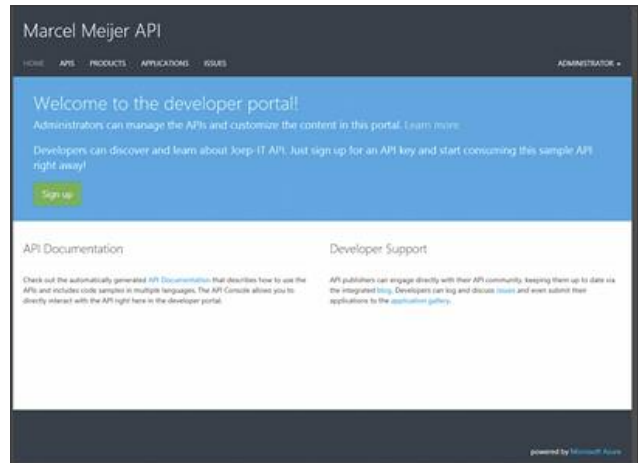
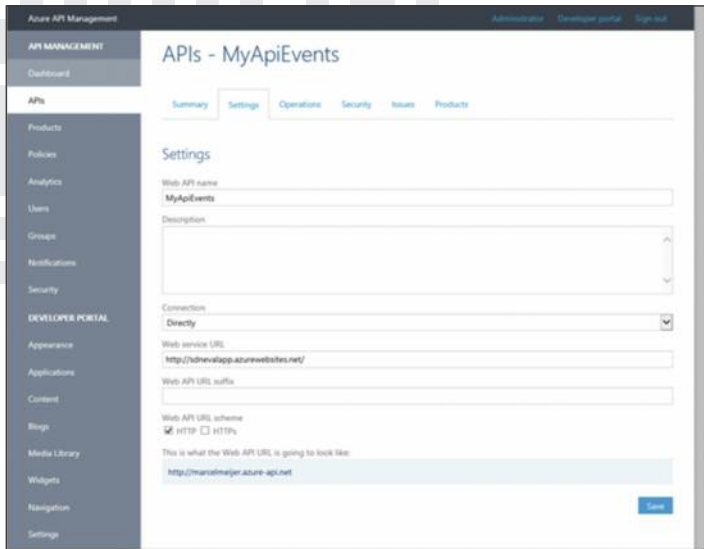
After creating the service, there is a different portal (<https://marcelmeijer.portal.azure-api.net/admin>) to do the settings, looking at the metrics, the applications, the usage etc.

But still all these solutions are far from optimal. The documentation is part of the source, which means redeploying an API after a documentation typo change. There is no monitoring and auditing on the API for individual users/developers/apps. If I give the API uri's to a developer which uses the API's in a very chatty way and makes it almost impossible for other developers to use my API. It is impossible to revoke the rights of a specific user on using the API.

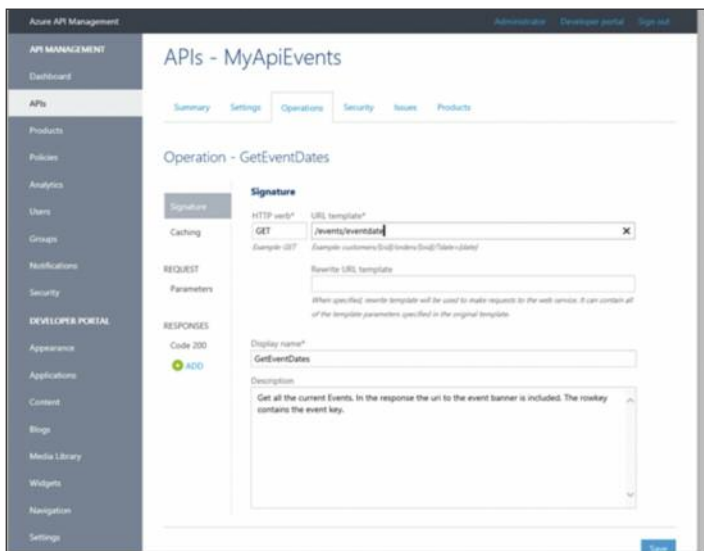
Luckily there is a solution on the Azure platform. The API management service (documentation site).

At the settings is the place to make the webservices available. The different operations, HTTP actions, the response codes, which URL, description and informational texts. If you have a Swagger doc or an ApiApp url, based on these an API can be imported without the manual labor.





There is a handy overview of the available API's.

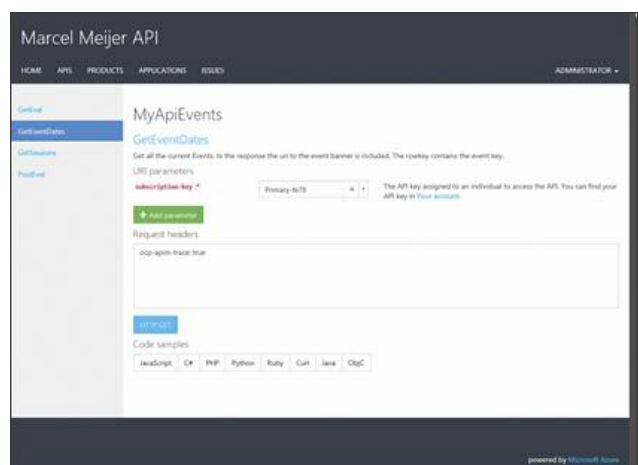


From the available API you can see the endpoints. You see the descriptions and the URI for calling the endpoint. To use the endpoint in an application the addition of a subscription key.

The URL to the source services can also be hosted on-premises. Of course it is smart and wise to secure the endpoints on this URL with Certificates, Username/Password combination or with OAuth

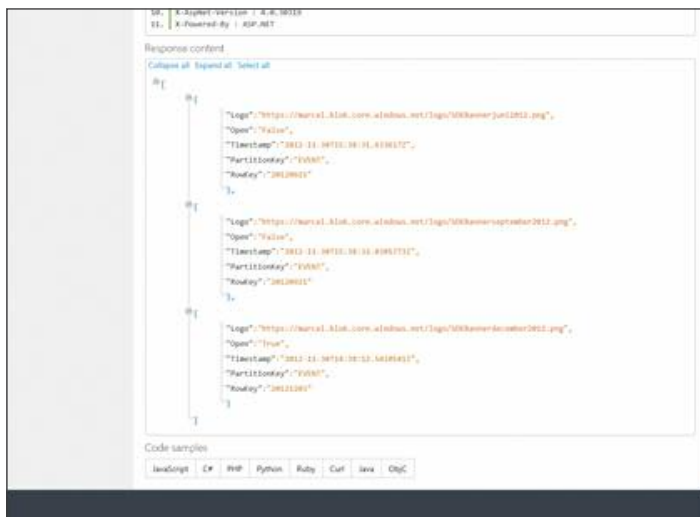
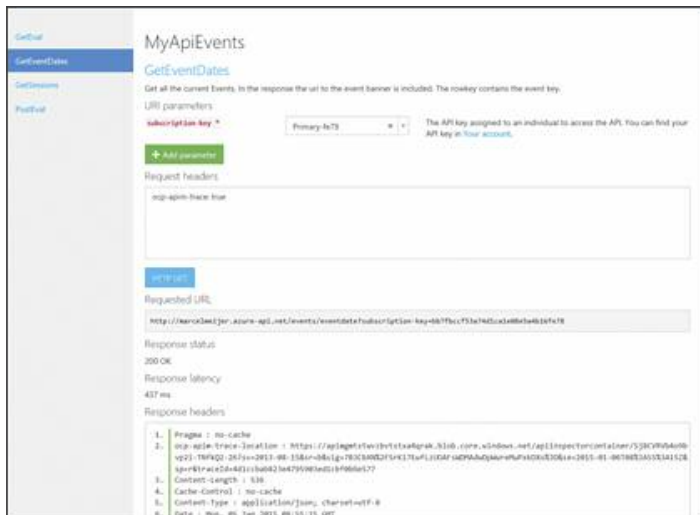
This was the management portal for the admin of the API (<https://marcelmeijer.portal.azure-api.net/admin>), for the developers there is a separate portal (<https://marcelmeijer.portal.azure-api.net/>). Which can be styled and changed within limits.

This Developer portal is rather complete. All the mentioned functionality can be found on it.



The whole idea of this portal is to regulate usage and with these subscription key is bound per API to an application/developer. Because the base endpoints are secured by Certificate, Username/Passwords or with OAuth, bypassing the API management is useless.

On this Developer portal there is even a possibility to try out the API method for the different HTTP actions. The trace and the result is shown.



At the bottom of the page you can find example code for a lot of programming languages. Everything to help your 'customers' of your API.



As I told in the beginning of this article, making an API available is one thing, but document/manage it is another thing. By using the Azure service you can focus on the fun and most important part of the API, the functionality.

An API isn't a hype anymore, but more used for business creation. Here the adage is: "build an API around your Business Model and not a Business Model around your API".

Using the specialized products for API management for third parties gives you more focus on your API and less on the management part. Why develop it yourself, when you can use the expertise of others. "You can reach further while standing on the shoulders of giants"



Marcel Meijer

Lead Consultant

<https://xpirit.com/specialists/marcel-meijer>



Think ahead. Act now.

Xpirit is the youngest member of 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.