XPRT.

Magazine N° 4/2017

Next Gen Cloud

Xpirit

Think ahead. Act now.

PROUDLY PART OF XEBIA GROUP

TechDays

Discover endless possibilities

2017

CREATE BUILD REINVENT EXPLORE

Take the high road and come to **TechDays 2017** to learn from best-in-class speakers and industry experts about how to reach new heights of technology which can best serve your business.

Does your business suffer from that most difficult of tasks; getting everyone on the same path? Then this must-attend event at the **RAI Amsterdam on the 12th & 13th October 2017** is for you! See how Microsoft technologies can combine with open-source and other technologies to move your **IT professionals, developers**, but more importantly, your entire team **to the next level** of collaboration.

www.techdays.nl Powered by GP Strategies Netherlands & Microsoft Netherlands

Microsoft

Colofon

XPRT. Magazine N° 4/2017

Editorial Office
Xpirit Netherlands BV

This magazine was made by
Vivian Andringa, Pascal Naber, René van Osnabrugge, Martijn van der Sijde, Loek Duys, Alex Thissen, Kees Verhaar, Geert van der Cruijsen, Chris van Sluijsveld, Marco Mansi, Cornell Knulst, Marcel de Vries, Pascal Greuter, Alex de Groot, Roy Cornelissen, Jesse Houwing, Sander Aernouts, Jasper Gilhuis, Marc Duiker, Peter Groenewegen

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

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

Translations
TechText

Gold
Microsoft Partner
Microsoft

docker
PARTNER

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

In this issue of **XPRT.** Magazine our experts share their knowledge about Next Gen Cloud

DEPT OF

"Willing, Able, and Ready to Change"

Working with Today's Trends. That's How We Roll at Xpirit!

We get most of our inspiration from the energy and passion of the people in our company, from creating new things together. It's all about the culture we have nurtured and fostered over the years. We presented a vision two-and-a-half years ago: we were going to do things completely differently. Our number-one priority is our employees: People First! There are few places as inspiring and dynamic as Xpirit. Our workforce isn't straitjacketed by job descriptions or a top-down hierarchy. We all have our own set of responsibilities, and we're really in this together. We take team spirit to a whole new level!

Author Vivian Andringa

One of our guiding principles is to remain focused on working with communities and sharing knowledge without expecting anything in return. That's why we are proud to present this fourth issue of *Xpirit Magazine*, which is about just that!

"Help us transition to the cloud," is a frequently heard request from our clients. The first step in this process tends to be making sure a clearly mapped-out procedure is in place. Some companies try to take the easy way out and go with the lift-and-shift approach, where in-house apps are replicated in the cloud without being redesigned (i.e., a bare-bones IaaS model). But they end up running into all kinds of architectural issues, while having little ability to make the required adjustments. This is where the cloud can become something of a poisoned chalice, doing more harm than good. That's why we always advise clients to start at the base and review the fundamental architecture. They need to think carefully about any re-architecting required and the integration of IT Operations. DevOps is fundamental to this process. You need to make sure your test automation and release management are in order before migrating to the cloud. As part of ALM, DevOps is the bedrock of a successful cloud migration – and it's crucial to maintaining a solid infrastructure going forward, with all the various technical steps involved.

Using Azure to smoothly transform IT Operations into Self-Service
As the ultimate self-service platform, Azure is able to accelerate DevOps. DevOps involves bringing together all the various IT disciplines, including development and IT operations. In order to accelerate the development and delivery of new solutions, you need a faster IT operations environment – preferably using a self-service model which allows developers and IT operations, working together, to provision the systems they require at that particular time. The bonus is that it's set up as a pay-as-you-go/pay-per-use system, giving you as much computing power as you need.

Is a full Cloud Migration really the smart thing to do?
Is it really more secure to store your data in the cloud than on a server somewhere? Public opinion on this is shifting from 'the cloud is scary' to 'the cloud is the default.' Increasingly, people feel there's no point having your own data center – after all, how can you guarantee the level of security public data centers are able to offer? It's a fascinating mind shift that's taking place. Cloud computing is no longer just something businesses will opt for on technical grounds: it's very much on people's radars. The cloud is growing in popularity among many home users. If these individuals feel it is safe, this will make them more accepting of cloud-based technology being used by banks, insurance providers and other service providers. Consequently, we are seeing a shift towards cloud solutions across the board.

Whether businesses will permanently shut down their data centers depends on the type of company and type of business involved. If the data is of a sensitive nature, there will probably always be room for a hybrid cloud system, where the data is divided between on-premise and cloud-hosted systems.

Thinking in terms of Disruptive Solutions
There are a number of businesses that regard IT as their main business driver. Financial service providers are essentially operating IT companies – as are banks, insurers, and utilities companies. These companies need to find new revenue models in the digital sphere, which means that the lines between their business operations and the IT backend are becoming increasingly blurred.

Large enterprises need to be aware that competition can come from unexpected sources. Start-ups and small spin-offs can land major accounts by luring them in with appealing propositions. Enterprises need to ask themselves: 'Are we flexible and adaptive enough to embrace this and focus on sharpening our competitive edge?' You no longer need to make investments to compete with major corporations: the combination of a sharp mind and an ingenious idea has the power to really make an impact. Many of today's established enterprises lack the kind of speed required to achieve change. Businesses have to start thinking more in terms of disruptive solutions.

IT companies need to learn to take a backseat. Many of them seem to think they know better than you do how your business should be run – but it's important to really work with the client and take a comprehensive approach. Take our client Van Lanschot, for example, a Dutch bank dating all the way back to 1737. Aware that they were losing out to their twenty-first-century counterparts, they decided they had no choice but to reinvent themselves as an omni-channel bank. This transformation incorporates IT, marketing, and business elements. We mapped out and managed the entire process for Van Lanschot – a process that requires a completely different type of IT consultant than 20 years ago.

"The half-life of your technical expertise is only going to shrink."

Pascal Greuter, Xpirit Managing Director

“Cloud done right is what we live and breathe! The business needs speed, cloud enables you to innovate at the speed of light, when done right!”

The Consultants of the future: Trusted Advisors at the Board Level

Can you brainstorm with clients at a high level? Are we the trusted advisors that board members need? How can we organize your IT operations down to the coding level? A modern-day software craftsman needs to be able to deliver a variety of knowhow, skills and techniques.

In addition to being a technical whiz, you need to be a strong communicator with a genuine vision. As part of our job assessment process, we test candidates to see if they dare to get out of their comfort zones. Being geared toward personal growth is key in this regard: allow yourself to be challenged and learn new things. Be innovative and persistent, and take on projects outside of your wheelhouse. Investing in expanding your skillset is crucial in this context – but to be successful, you also need to have the right mindset. “Rather than getting hung up on what you might lose, you should see change as an opportunity to try something new.”

Innovation is about trying out new things without rushing back to your comfort zone when the going gets tough. It requires persistence – and that’s where you’ll often find that people are stuck in their ways.

“The half-life of your technical expertise is only going to shrink.” The knowhow you possess now will be largely obsolete just a few years from now. You need to remain aware that, while you may be able to call yourself a specialist today, a couple of years down the line you’ll need to be attuned to, and understand, the trends of that moment. You need to embrace that constant state of flux, which is why we make a point of investing so much in knowledge development. It’s vital that our consultants keep learning all about the latest and greatest new things, since we know that in two years’ time what counts as state-of-the-art now will be old news.

This magazine was developed with this in mind. It aims to keep you up-to-date on some of the most exciting developments happening in IT right now. Be sure to pass it on! </>

Marcel de Vries, Xpirit CTO

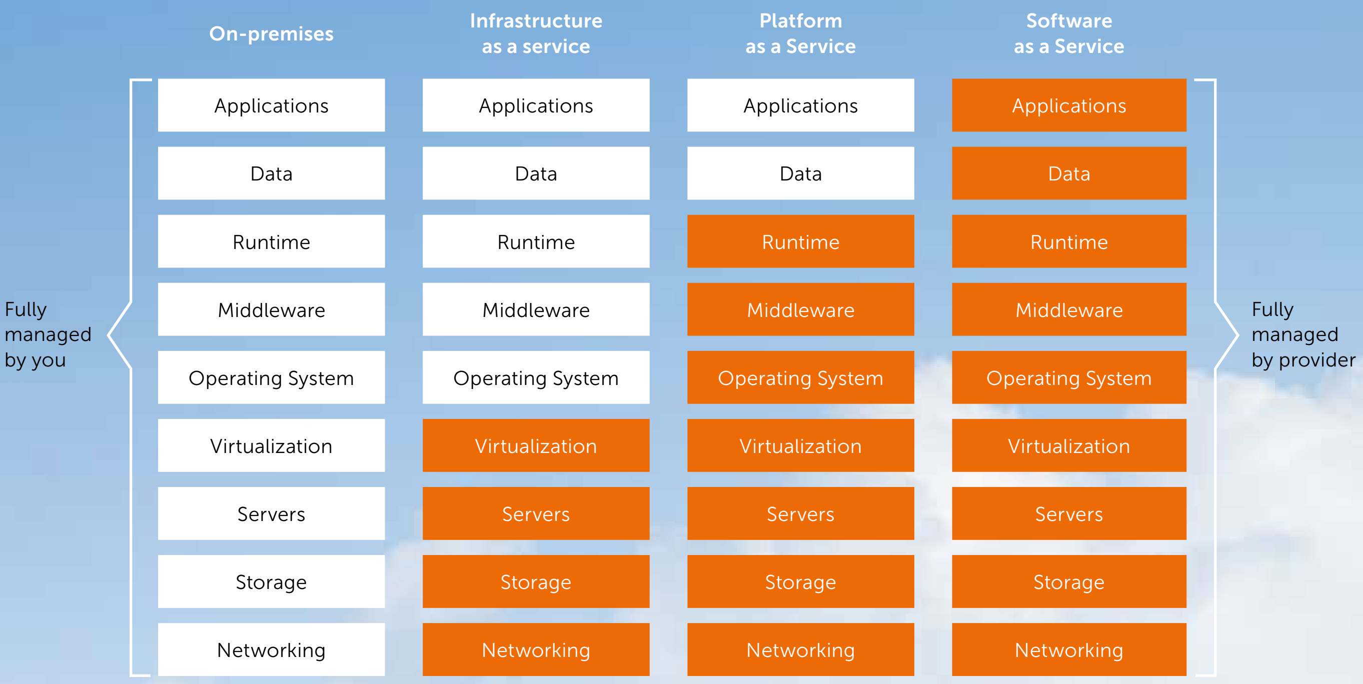
The (r)evolution of Cloud Computing

If there is one term that stands out in the current list of technology developments, trends and buzzwords, it is serverless computing. Serverless computing is the next step in the evolution of cloud computing (delivery of computing services – servers, storage, databases, networking, software, analytics, and more – over the Internet). Serverless computing takes away the necessity for you to think about the computing capacity required for running your software. And what's more: it lets you execute and pay for it on-demand. What is serverless computing exactly and what are the benefits?

Author Martijn van der Sijde

Summary of Cloud Service Models

Before we discuss the concept of serverless, have a look at the following diagram that presents a short recap of Cloud Service Models. As you go through each of the diagram's pillars from left to right, the focus shifts to the activities required to provide application functionality to end-users.



On-premises	Infrastructure as a service	Platform as a Service	Software as a Service
⬜ Applications	⬜ Applications	⬜ Applications	🟧 Applications
⬜ Data	⬜ Data	⬜ Data	🟧 Data
⬜ Runtime	⬜ Runtime	🟧 Runtime	🟧 Runtime
⬜ Middleware	⬜ Middleware	🟧 Middleware	🟧 Middleware
⬜ Operating System	⬜ Operating System	🟧 Operating System	🟧 Operating System
⬜ Virtualization	🟧 Virtualization	🟧 Virtualization	🟧 Virtualization
⬜ Servers	🟧 Servers	🟧 Servers	🟧 Servers
⬜ Storage	🟧 Storage	🟧 Storage	🟧 Storage
⬜ Networking	🟧 Networking	🟧 Networking	🟧 Networking

⬜ Managed by you 🟧 Managed by your provider

Overview of Cloud Service Models.

Martijn van der Sijde

As an architect, Martijn helps organizations translate their business strategy and challenges into technical solutions in close collaboration with the people involved. Martijn leverages his vision and knowledge on cloud-native architectures, ALM, and security to design and implement modern architectures, including the required changes on people, processes and technology. He has a passion for learning and applying a (technical) improvement or innovation to achieve an increase in the delivery of customer value.

The model that is closest to on-premise IT is Infrastructure as a Service (IaaS). This model provides the basic building blocks for Cloud Computing.
The dedicated or virtualized hardware (networking, storage and computing resources) is owned and hosted by the Cloud Service Provider and is provided to a company in a virtual manner.
This company can self-provision the infrastructure on-demand, and does not have to worry about maintaining the hardware. IaaS allows the company to focus on utilizing the infrastructure and consumption-based payment, instead of maintaining the infrastructure and making investments on it.

IaaS allows the company to focus on utilizing the infrastructure and consumption-based payment.

The second service model, Platform as a Service (PaaS), provides the operating system, middleware and runtime on top of the infrastructure layer. PaaS allows companies to focus on managing the scale of the infrastructure, in addition to the deployment and management of their applications. Management of underlying infrastructure is abstracted away.

PaaS allows companies to focus on the deployment and management of their applications.

The last model in the picture is the *Software as a Service (SaaS)* model, in which the application and data are also managed by the provider. The software is licensed on a subscription basis and is hosted centrally. Well known examples of these complete software products, available as SaaS services, are: Office365, Google Apps, and Sales-Force. SaaS allows a company to focus on how to use the functionality provided by the application, while it does not have to manage feature additions, servers, or operating systems.

Serverless computing

Now that we understand the various cloud service models, we can look at the position of the concept of serverless computing. It should not come as a surprise that serverless computing does not mean that there are no servers involved. The core idea of serverless computing is that you don't have to think about the infrastructure and computing capacity required for your logic. Answering questions like how many instances of your application are required or how many Virtual Machines you need become obsolete. In this sense, the "serverless execution model" can be positioned between the pillars PaaS and SaaS.

Serverless computing: Backend as a Service

The serverless concept can appear in different forms. One way to look at it is the use of 3rd party backend services within your own solution. Let's take an example by looking at identity provider functionality. Functionalities such as authenticating a user based on credentials and resetting a user's password are provided by products such as Azure Active Directory B2C, or Auth0. A functionality is executed on the basis of incoming events (http service calls). These products take care of scalability and let you pay per authentication.
In addition to this, some products allow you to add custom logic to their solution to tailor functionality to your needs. The functionality in itself does not have any end-user value – it is a "semi-finished" product that needs to be integrated in a solution. This appearance of serverless computing is referred to as *Backend as a Service (BaaS).*

Serverless computing: Function as a Service

Another appearance of serverless computing is about breaking down an application or a microservice into discrete functions. These small bits of code, that are configured to be executed on the basis of events, enable efficient resource utilization. The code of these functions is deployed and configured into a cloud environment, and the cloud provider takes care of running these pieces of code triggered by events coming in. This is also referred to as *Function as a Service (FaaS).* FaaS allows a company to focus on the logic that a piece of software needs to provide, and to pay for it per execution. In the case of FaaS, the focus is on the development and deployment of small pieces of source code.
Cloud providers have jumped in the FaaS space with their own offerings: AWS Lambda (Amazon), Azure Functions (Microsoft), Google Cloud Functions, and IBM OpenWhisk.

With FaaS the focus is fully on the logic that needs to be provided: the code.

FaaS matches perfectly with the concept of microservices, mobile and Internet of Things (IoT). It provides auto-scaling and load balancing out-of-the-box, saving you from having to manually configure clusters, load balancers, etc. The only thing you need to do is to give the code to your cloud provider and trigger it through events.

Characteristics of serverless computing

Giving a single definition for serverless computing is likely to lead to some discussion. The best way to define serverless computing is, therefore, to state the characteristics it complies to. If we look at the descriptions of BaaS and FaaS as described above, these characteristics are:

› it is event-driven
› computing is done on-demand
› scaling is done out-of-the-box

Events Execution of backend functionality or function code is triggered by an event. The types of events depend on the ones offered by the cloud provider. This can range from a file update on Amazon S3 or OneDrive, a timer event, a message on a queue, or an incoming HTTP request.

Containers Containers are used in the core. Containers wrap a piece of software in a complete file system that contains everything that is required to run this software: code, runtime, system tools, and system libraries – anything that can be installed on a server.
This guarantees that the software will always run in the same way, regardless of its environment.

With serverless computing, the application code is taken, placed into a container, executed, and torn down without you knowing anything about this process and its underpinnings.
This provides for execution on-demand and easy scaling.
The cloud provider takes care of finding a server for the code to run on, and it scales up if necessary. The way in which this is physically implemented, for instance the container architecture, varies between the cloud providers.

Benefits of serverless computing

Efficiency in IT spend FaaS, BaaS or other means of serverless computing can be much cheaper, because you pay per execution, and you don't pay for resources that are idle. What's more, no money is lost on managing the infrastructure and platform required to do scaling. There's a variety of pricing models available, so calculating the break-even point based on your demands is advisable.

Value-driven development Value-driven development lets you focus on the functionality that is to be delivered, and maximizes the time you spend on delivering true added value to your end-users. Non-value activities such as managing the infrastructure are removed from the software development process.
This fulfils the goal of Lean software development: eliminating waste by removing the non-value added components from the software development process, which is more cost-efficient.

Event-driven architecture As mentioned earlier: FaaS matches perfectly with microservices, mobile and IoT, which are the modern architectures of today. This is because of the event-driven nature and level of granularity of FaaS. This type of architecture with its loose coupling of components and good distribution is a real strength, allowing for architectural agility and high scalability.

A revolution?

Serverless computing is a logical evolution of Cloud Service Models.
In its current form I don't regard it as a revolution. Revolutionary would be the possibility of moving beyond virtual machines and containers. Imagine that you just upload your cross-platform application code, in your preferred programming language, to your favorite cloud provider. And your code will run without you knowing anything about the application container, execution environment, and the operating system it will be running on. It auto-scales out-of-the box and you pay per execution. That is going to be real serverless computing. </>

ACADEMY
TRAINING.XEBIA.COM
Xebia

Empower yourself.

Xpirit teams up with Xebia Academy to provide a variety of professional training sessions tailored to take you further.

No matter where you are on your professional journey, Xebia Academy's customized, world-class training gives you the skills and knowledge you need to advance and achieve more. Our collaborative, continuous learning approach and in-depth curriculum prepare you to take on today's latest challenges in Microsoft Application Lifecycle Management, Cloud and more.

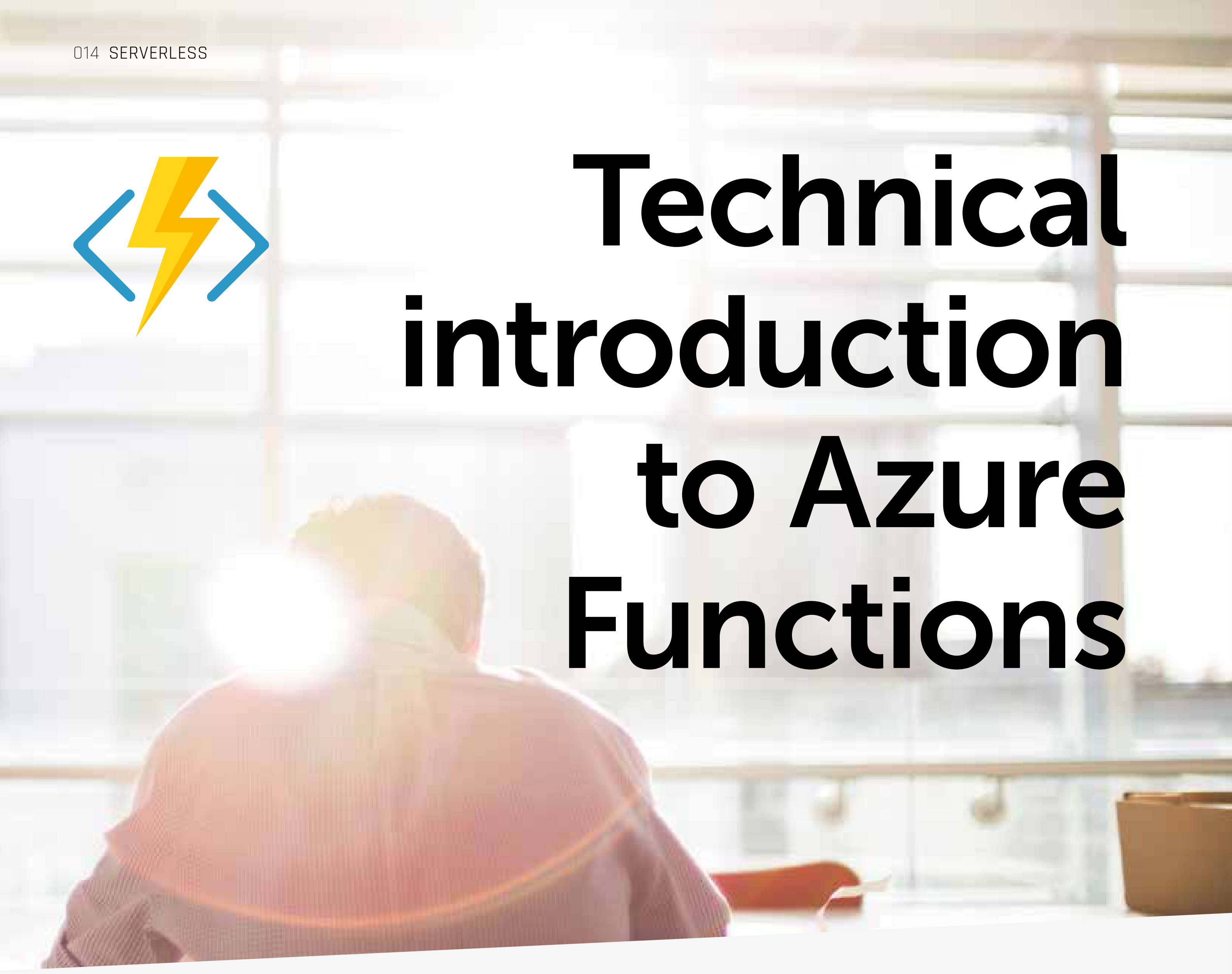
Take one of our Microsoft Technology public classes held at Xebia Academy state-of-the-art training facilities in Amsterdam, or learn with your colleagues in a customized, in-company training program, facilitated on-site at your location, anywhere in the world.

Visit Xpirit.com/training or training.xebia.com for more details.

Xpirit

training.xebia.com

Xebia Nederland B.V. • Laapersveld 27 • 1213 VB Hilversum • +31 35 538 19 21 • training@xebia.com
Amsterdam Office • Wibautstraat 200 • 1091 GS Amsterdam



Technical introduction to Azure Functions

Azure Functions is Microsoft's event-driven, serverless, cloud platform for creating lightweight background processes. It is Microsoft's FaaS (Function as a Service) answer to Amazon's AWS Lambda. A function is triggered by an event and many event triggers are built in the runtime. Azure Functions can be written in multiple programming languages.

Author Pascal Naber

Background
In November 2014 Amazon introduced AWS Lambda. This service made Amazon into the first major cloud provider with a serverless offering. Besides Amazon, there are various other vendors of serverless offerings. Google is working on Google Cloud Functions, which has not been released yet. IBM offers a serverless platform within IBM Bluemix, which is called IBM OpenWhisk. In March 2016, Microsoft announced Azure Functions on Build, and Microsoft already released Azure Functions with General Availability (GA) as early as November 2016.

In Detail
Despite the term serverless, Azure Functions naturally needs servers to run on. The term serverless refers to the fact that you don't need to provision or create virtual servers yourself to let the code run on. All necessary provisioning is done by Azure.
When you start with Azure Functions, you create a Functions app in the Azure Portal. This is the host of multiple functions, while a function app is a special kind of App Service: the portal provides a user interface to create functions and configures the triggers for it. It's also possible to set up Continuous Integration with Azure Functions.

When you start with Azure Functions, you create a Functions app in the Azure Portal.

Azure functions are based on the Azure WebJobs SDK. The Azure WebJobs SDK offers triggers, bindings and a runtime. Triggers are events that trigger your function. Bindings are declared in metadata and connect external resources to your function as trigger, input or output parameters. Azure Functions offers a layer on top of this. This layer is open source and can be found on github. (https://xpir.it/mag4-func1)

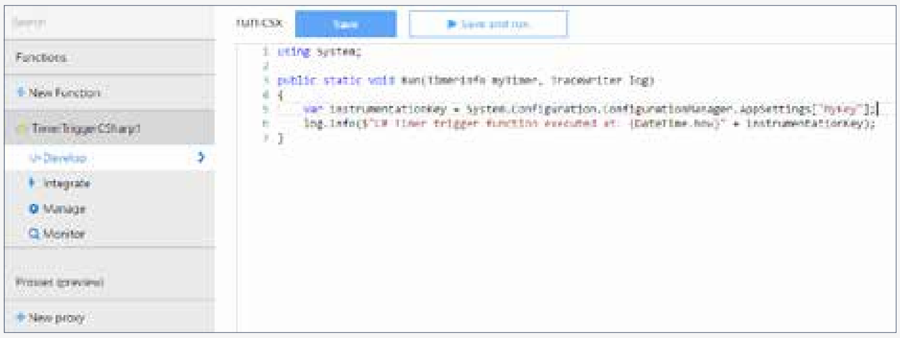
To create an Azure Function, you can choose one of the following programming languages: C#, PowerShell, Batch, Python, Bash, JavaScript, PHP or F#. When you choose a C# based function, the function is created in a csx file, which is a C# script file. When the function is executed for the first time, the script is compiled and executed in memory. You can compare an Azure Function with a method in a C# class. When you want to add other classes, these classes should be coded in the same csx file. It is not possible to separate files to put your code in.
A result of this is that you have to think about what you automate in your Azure function and the amount of code that is needed for this.
You should only have the code for the main process in your Azure Function to keep it maintainable. If you like, you can add references or Nuget packages to make use of Domain Models or other classes.

Pricing models

When you create an Azure Functions app, you can choose between two service plans: Consumption plan and App Service Plan.
The consumption plan is new and unique for Azure Functions. You only pay for the number of requests and the Gigabytes per second (GB-s).
GB-s stands for the time required for processing, multiplied by the allocated memory. The price includes 1 million executions or 400.000 GB-s.
The consumption plan scales the CPU and memory automatically up to 1.5 GB. It's possible to configure a daily quota to create a limit on the spending.
The App Service Plan is the same Azure Resource you use to host your webapps or webjobs, for example a Standard S2. You pay for the entire service plan with fixed expenses, and you are in control of the scalability, which can have a fixed or automatic setup.
In addition to paying for Azure functions, you also need to pay for a Storage Account. All function types except HTTP triggers require a storage account.

Adding a function

After adding an Azure Function in the Azure Portal, the first thing you need to do is to choose the language you want to create the function in. After that, you choose how the function will be triggered, based on a predefined template (See all triggers in the table below). The last step consists of configuring the settings belonging to the selected trigger. For example, when you choose the Event Hub binding, you have to configure the access to the Event Hub. After the function has been added, the edito in the Azure portal offers functionality to develop, run and monitor the function.



```
using System;

public static void Run(TimerInfo myTimer, TraceWriter log)
{
    var instrumentationkey = System.Configuration.ConfigurationManager.AppSettings["mykey"];
    log.Info($"C# Timer trigger function executed at: {DateTime.Now}" + instrumentationkey);
}
```

Development environment in the Azure Portal.

Features

Azure Functions can be started on the basis of a trigger (event). Azure Functions can also integrate with other Azure Services, which is called bindings. The substantial power of Azure Functions becomes manifest through the integration with other Azure PaaS resources such as ServiceBus, Storage, and EventHub. The following table shows which triggers and bindings are supported with Azure Functions:

Binding	Trigger	Input	Output
Function app Schedule Triggers based on a schedule which is configured with a CRON expression.	●		
Http (REST or Webhook) Invokes a function with an Http Request, responds to webhooks and allows to respond to requests. Like a GitHub webhook.	●		●
Blob Storage Triggers for new and updated blobs in the container, read blobs or write blobs.	●	●	●
Event Hub Event Responds to an event send to the Event Hub or sends a message to the Event Hub.	●		●
Storage Queue Monitors a queue for new messages, responds to them and writes messages to a storage queue.	●		●
Service Bus Queue & Topic Responds to messages from a queue or topic and creates a queue message.	●		●
Table Storage Stores read tables and writes entities		●	●
Mobile App Storage Reads from and writes to data tables.		●	●
Document DB Reads or writes a document.		●	●
Notification Hub Push notification Sends push notifications.			●
Twilio SMS Sends an SMS text message.			●

A trigger can be the event that starts the Azure Function. Please note that not all bindings support a trigger implementation. For example, it is not possible to start your Azure Function based on the event of a new document in DocumentDB. An Azure Function can only have one trigger and an unlimited number of inputs and outputs.
Bindings can be configured in the Azure Portal in a user-friendly user interface. The bindings are stored in the function.json file. This file can be edited in the Azure Portal also.

The binding for a trigger for an Event-Hub looks like this:

```
{
 "bindings": [
   {
     "type": "eventHubTrigger",
     "name": "myEventHubMessage",
     "direction": "in",
     "path": "myeventhub",
     "connection": "ehconnection"
   }
 ],
 "disabled": false
}
```

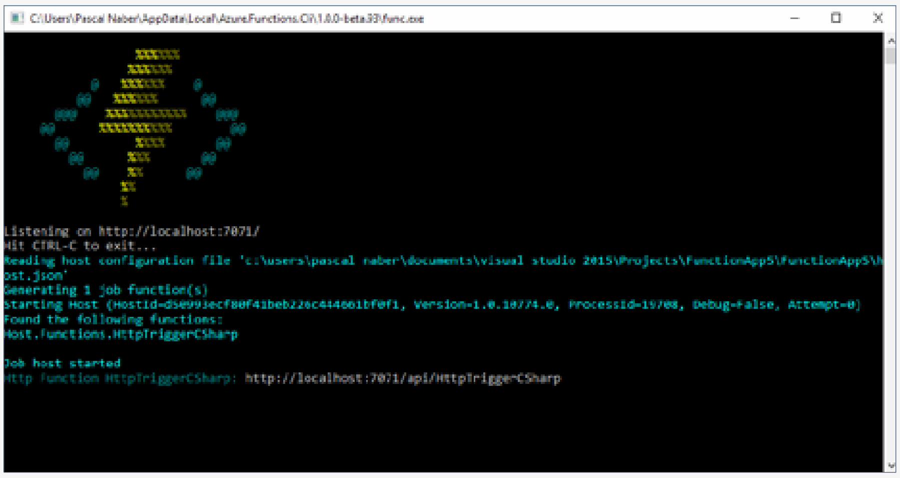
Local development

Since December 2016 it has been possible to develop functions locally in Visual Studio. To make this possible in Visual Studio you have to install The Visual Studio Tools for Azure Functions.
https://xpir.it/mag4-func2

At the time of writing this article, this functionality is only available for Visual Studio 2015 update 3 and as yet, there is no version for Visual Studio 2017.
The tools are a prerelease version and there are limitations. The most striking limitation is the limited support for IntelliSense in csx files. For example, there is no support from IntelliSense for input parameters.

The Azure Tools are offering a new Visual Studio Project template called Azure functions (prerelease). A project can contain multiple functions.
The tooling also offers the same function templates as the Azure portal.
Another major benefit of the tools is that it is possible to run Azure Functions locally and debug them.

When you debug a function for the first time, Visual Studio will ask you to download the Azure Function CLI. This CLI is needed to debug the Azure Function and acts locally as the host for the function. When an update is available, Visual Studio will ask you to download the latest version.

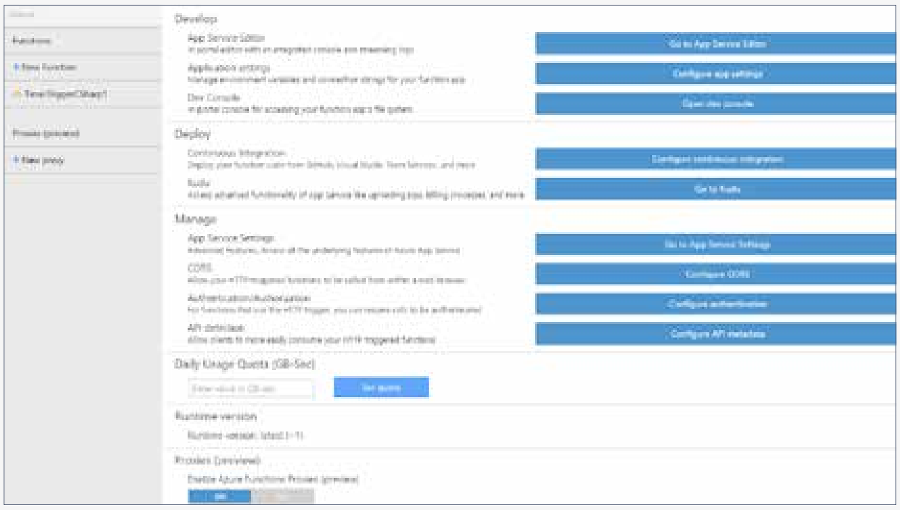


```
C:\Users\Pascal Naber\AppData\Local\Azure.Functions.Cli\1.0.0-beta.93\func.exe

Listening on http://localhost:7071/
Hit CTRL-C to exit...
Reading host configuration file 'c:\users\pascal naber\documents\visual studio 2015\Projects\FunctionApp5\FunctionApp5\h
ost.json'
Generating 1 job function(s)
Starting Host (HostId=d5e993ecf80f41beb226c444661bf0f1, Version=1.0.10774.0, ProcessId=19708, Debug=False, Attempt=0)
Found the following functions:
Host.Functions.HttpTriggerCSharp

Job host started
Http Function HttpTriggerCSharp: http://localhost:7071/api/HttpTriggerCSharp
```

Running an Azure function locally with the Azure Function Tools.

Settings

AppSettings and Connectionstrings work in the same way as Web Apps and Web Jobs. Similarly, the settings are stored in the application service. In the Azure Function Tools, appsettings and connectionstrings need to be added in the appsettings.json.



Settings on an Azure Function app.

References

Developing csx files is different from the normal way of developing C# classes. For example, it is not possible to set assembly references by setting a reference in the same way as you used to in Visual Studio.
To set dependencies to libraries it is possible to reference .Net 4.6 Nuget packages. These references are stored in the project.json file.

After that, you add the following line to the csx file:

```
#r "Microsoft.Azure.WebJobs"
```

It is also possible to reference your own assemblies. Add the assembly to the bin directory of the Azure Function and add a reference with the following line:

```
#r "MyAssemblyName.dll"
```

Proxies

The latest feature at the time of writing this article is Azure Function Proxies. Proxies allow you to combine multiple Azure Functions in one large API. This large API is a façade, i.e. a single point of entry for the outside world and it forwards the calls to other Azure Functions. It is the light version of API Management, without throttling, security, and caching, with the benefit that it is cheaper.

Testing

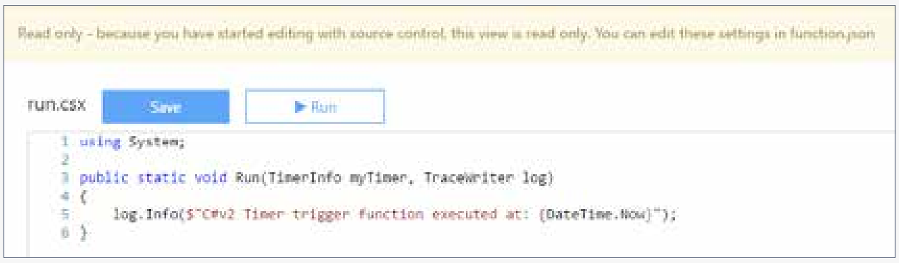
With ScriptCS (csx files) it is not possible to unit test Azure Functions in Visual Studio. The functions can only be tested when the function runs in Azure.

Precompiled functions

One solution to make unit testing of functions easier is to use precompiled functions. With precompiled functions, you are actually developing the usual C# code instead of C# scripting code. This code is compiled into an assembly in the way you are used to.
The compiled code can be deployed to Azure as an Azure Function. To do this, choose a Class Library project instead of an Azure Function project. This code can be tested in the same way you are used to. To let it behave as an Azure Function you need to add a couple of NuGet references, and you need to add a function.json configuration file.
A disadvantage of the precompiled function way of working is that you cannot edit the code in the Azure Portal anymore, because the file is in a binary format.
If you want to know more about this, please read the blogpost of our colleague Geert van der Cruijsen https://xpir.it/mag4-func3

Continuous Deployment

By default, the functions in the Azure Portal are not under source control. However, there are a number of options available to create integration with source control. Azure functions provide functionality to configure Continuous Deployment with several source code providers such as GitHub and VSTS. After configuration, the Azure Functions will be read-only in the Azure Portal.


Read only - because you have started editing with source control, this view is read only. You can edit these settings in function.json

```
using System;

public static void Run(TimerInfo myTimer, TraceWriter log)
{
    log.Info($"C#v2 Timer trigger function executed at: {DateTime.Now}");
}
```

An Azure function becomes read only after configuring source control integration.

After each commit the entire Azure function app will be updated. However, you need to use separate branches if you want to be in control when a functi-on app is deployed.

Another possibility is to use a Build and Release pipeline in VSTS for example. You can read how to do this in the blogpost of our colleague Peter Groenewegen: https://xpir.it/mag4-func4

Conclusion

From a technical viewpoint, you can program the same functionality in C# with an Azure Function as with an Azure WebJob. One of the reasons why Azure Functions is interesting is the dynamic pricing model. You don't have to pay any attention to any infrastructure dependency because you only pay for what you use and the first one million calls are free. In addition, the automatic scaling of pricing model is a powerful feature, which is another aspect that saves costs and makes your life easy. A final advantage of Azure Functions is that you don't need to have an IDE (Visual Studio) to develop your functions. You only need a web browser!
Azure Functions force you to create small, self-contained pieces of functionality, which are event-driven and can be updated separately.
These are characteristics of Micro services also. If you can handle the way you manage the source code, tests and updates this could be a neat approach. My experience is that Azure Functions already prove their power when using them to automate small separate processes like checking resources based on a timer or a different trigger. Azure Functions provides you with more agility because you only have to consider business logic and not infrastructure. The Microsoft Azure Function team constantly updates the features for Azure Functions, so new functionality is on its way. </>

Pascal Naber
Microsoft Azure MVP &
Professional Scrum Master

Pascal helps companies embrace Microsoft Azure and build large scale distributed systems with modern architectures based on microservices. He is the co-founder of the Dutch Azure Meetup, for which Microsoft has awarded him with the Microsoft Azure MVP award. In his spare time, he enjoys killing monoliths just for fun.

#ddsw17

June 6-9/2017

Amsterdam / Huizen
Nieuwegein / Rotterdam
Zeist

Dutch Data Science Week

From June 6 – 9 the Dutch Data Science Week will take place across The Netherlands with the purpose to provide Data Scientists and organizations with a platform for innovation, to learn, inspire and network. Expect a wide range of activities, including conferences, hackathons, training classes, workshops and meetups.

- Grow your Data Science skills with a training class or workshop
- Hack away during one of the challenges
- Meet interesting organizations at the career café
- Be inspired during a meetup or conference

The Dutch Data Science Week is a joint initiative of SAS Netherlands and GoDataDriven in close collaboration with various partners and supporters.

www.dutchdatascienceweek.nl

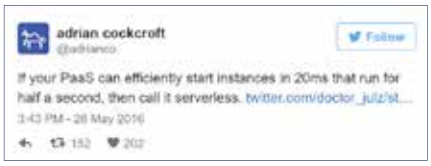
The Serverless lifecycle: is it really that different?

In today's world, technological innovation is moving at breakneck speed. It was only 15 years ago that we were deploying applications on bare metal servers using floppy disks. Since then, we have moved to virtual machines, Infrastructure as a Service (IaaS), Platform as a Service (PaaS), Software as a Service (SaaS). And recently we've moved into a world where container technology is starting to become mainstream. The next step is serverless computing. Or, as Martin Fowler[1] describes this in his blog, Functions as a Service (FaaS) and Backend as a Service (BaaS). This is yet another "*aaS" to keep in mind when selecting your hosting platform. And just like any other application, a serverless application requires proper lifecycle management. In this article, we'll discuss this serverless lifecycle, and see where it differs from "normal" applications that you all know.

Authors René van Osnabrugge and Kees Verhaar

Serverless computing

So what is serverless computing?
Of course there is still a server involved somewhere. It's just not managed by you, but by a cloud provider. For a thorough explanation, please read the article The (r)evolution of Cloud Computing in this magazine (see page 010). As far as this article is concerned, we think that serverless can best be seen as PaaS++. The following tweet from Adrian Cockroft describes it very simply:


adrian cockcroft
@adrianco

If your PaaS can efficiently start instances in 20ms that run for half a second, then call it serverless. twitter.com/doctor_julz/st...

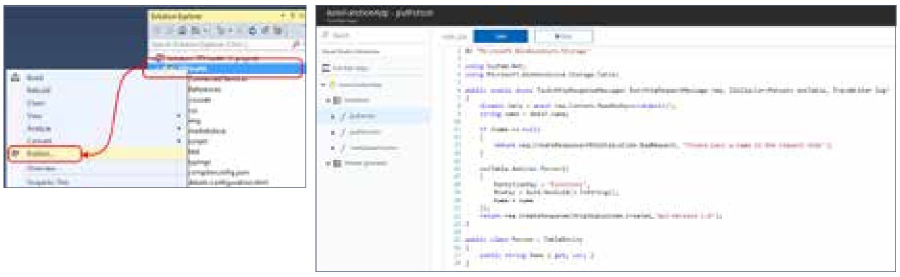
3:43 PM - 28 May 2016

152 202

When it comes to FaaS, both Amazon and Microsoft offer cloud-based solutions. Amazon Web Services (AWS) offers AWS Lambda and Microsoft's Azure offers Azure Functions.
This article primarily focuses on the Azure stack, but it also applies to AWS.

The Serverless Lifecycle

A serverless application is usually very small, but nevertheless it is still an application. And as already mentioned, every application has a lifecycle and this lifecycle needs to be managed. The FaaS implementation in Azure, Azure Functions, is fairly new, and the marketing around Azure Functions focuses a lot on the easy and friction-free editing experience "in the browser". Although this is very powerful, it is also very dangerous and it may even be unwanted when it comes to a production application. Just like the infamous "Right-click, Publish" experience in Visual Studio, this lets you publish your application straight to production without following any process at all. Of course, there are rights and security, but still, imagine what happens when someone "accidentally" edits a heavily used production function.



The infamous "Right-click, Publish" experience in Visual Studio, and the Azure Functions equivalent to editing in the browser.

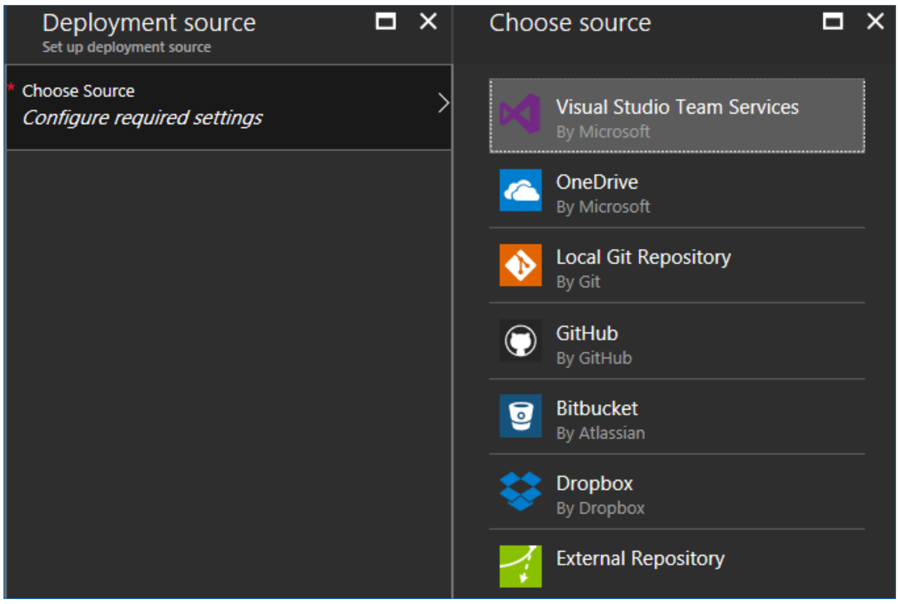
[1] https://xpir.it/mag4-slc1

It is important to treat a serverless application in the same way as any other application. The following diagram shows a typical application lifecycle. In the following paragraphs, we will apply each of the phases from the diagram to a serverless application.

Requirements Will requirements change for serverless applications? Surely it depends on how you define your requirements, but requirements should describe the functional and non-functional part of an application, while staying away from technical implementation details. A serverless architecture deals with the technical implementation of a requirement. The process of gathering and refining requirements itself should not change.

Development The impact on your development cycle is probably higher. There are a lot of choices to be made up-front. Which language are you going to use? Azure Functions supports many languages like C#, Python, Node.JS, Powershell, PHP, etc. And which IDE are you going to use? The most important thing is to think ahead and define your test, build and release strategy to make a good choice. For example, if you want unit tests as well as a build and a release pipeline, editing your code directly in a web browser is probably not the best option. Let's have a look at our development strategy and the options you have.

What are we developing? When you build a serverless app, you can download a project template in Visual Studio and get started. However, in most cases this is not enough. Your application (or should we say function) probably needs a backend, a data store, Azure Blob Storage or a MongoDB. You can create this manually and configure your function accordingly, but in the modern DevOps world we should strive to automate everything. This means that you should develop the infrastructure for your application together with that application itself, using the infrastructure-as-code paradigm to enable fully automated deployment. Whether you're using an ARM template on Azure, CloudFormation on AWS, or Docker containers on a hosted cluster, in each case you need to develop both the application code and the infrastructure code.


Deployment source
Set up deployment source
* Choose Source
Configure required settings

Choose source
Visual Studio Team Services
By Microsoft
OneDrive
By Microsoft
Local Git Repository
By Git
GitHub
By GitHub
Bitbucket
By Atlassian
Dropbox
By Dropbox
External Repository

How to develop And then there is the question of how you are going to develop your code. Focusing on Azure Functions as a platform with C# as your language of choice means that you have four options to develop serverless applications.

Write code directly in the browser
The fastest way to create an Azure Function is to write, test and monitor it directly in the browser. However, ease of use comes at a cost. You don't have a pipeline, unit tests or even a backing source control repository, so it is probably not suitable for your enterprise application, where a higher level of traceability and control is usually required.
However, creating and editing directly in the browser is great for rapid prototyping, quickly trying out an idea, or creating a small application that you'll use only for yourself.

Connect a repository to Azure Functions Using Azure Functions, you can connect an existing repository to your Azure Function, resolving the source control issue. You have various options for the type of repository that you want to use, for instance a repository in VSTS or GitHub, but solutions like OneDrive and Dropbox are also an option.

Whenever a change is committed or saved in the repository, this will be deployed automatically. In the case of OneDrive or Dropbox, this happens whenever somebody changes a file. In case of a Git repository, deployment is always connected to a specific branch (e.g. master). This gives you a bit more control, since you can use branch policies[2] to control which code is merged into that branch. But still, for larger organizations where an application typically goes through different stages of testing and acceptance before ending up in production, this does not provide the required level of control. This mode of development & deployment is most suited for small teams that write small applications and don't need the additional verification steps provided by a build and release pipeline.

Using your favorite IDE and its capabilities If you want to step it up a notch, you could make use of the power of your favorite IDE. This means that you can run your Azure Functions locally, allowing you to debug them. And if you develop them like any regular application, you can use familiar practices to get things like traceability and controlled releases in place.
Code is stored in source control and you can create a build & release pipeline (see Technical Introduction to Microsoft Azure Functions) to publish your functions to Azure.
When you write Azure Functions in C#, you should be familiar with the Visual

[2] https://xpir.it/mag4-slc2

Studio Tools for Azure Functions. They let you create an Azure Functions project inside Visual Studio, with some nice boilerplate code already in place. The only challenge that remains in this scenario is testing. Testing your Azure functions written as a C# Script File (CSX) is quite limited. The Visual Studio Tools for Azure Functions will give you the local compile, build & debug experience with things like breakpoints, watches, etcetera. However, running unit tests on CSX files is not possible.
This means that most of your local testing will be manual, which is not sustainable in today's world, where the focus of your testing should be at the unit test level (The Test Pyramid concept: https://martinfowler.com/bliki/TestPyramid.html).
Because of this limitation, we recommend using the precompiled assembly's approach, as described in the following paragraph.
However, Azure Functions supports quite a few other languages (like Node.js and PHP) for which unit testing is very well possible. If you're using any of those, then the Azure Functions CLI[3] will let you run and debug your Azure Function locally[4].
Using this approach allows you to develop enterprise grade applications using Azure Functions.

Write CSX files and reference a C# class library When you're writing C# and want to stay close to the usual development flow, the best alternative is to use a precompiled assembly that you can reference in your CSX file, leaving your CSX as nothing more than a wrapper around a "normal" class library[5]. When you use the class library, you can use the default toolset for Unit Testing and all other features you are used to in Visual Studio, storing things in Source Control, creating builds, releases, and still have a state-of-the-art serverless function.

The following table summarizes the options for development as outlined before:

	Source control	Testing	Deployment	Typical application
Write code directly in the browser	None	Manual testing using e.g. Postman or curl	Code is published directly when you click "Save"	Rapid prototyping
Connect a repository to Azure Functions	Basic (e.g. One-Drive) to full (e.g. Git)	Possible, but no integration	Straight from the repository to production	Small teams with no need for build & release pipeline
Using your favorite IDE and its capabilities	Full	C#/CSX: No unit testing, other languages: full	Full build & release pipeline	Enterprise scale applications
Write CSX files and reference a C# class library	Full	Full	Full build & release pipeline	Enterprise scale applications

Build

The purpose of a build pipeline is to produce artifacts that can be deployed throughout multiple environments. For example, when you create a normal web application, you build an MSDeploy package that you provide with the right values for its parameters in your release pipeline. Or, when you use containers, your build pipeline will turn into a "bakery" to produce a container and publish it to a container repository so it can be used on any hosting platform that supports containers.
When it comes to your serverless application, your build will be the same as what you are used to. The build produces the binaries and other files for your application, the artifacts to create the underlying infrastructure, and it will have some configuration settings that can be modified during the deployment in your release pipeline.
Performing tests that do not require a running application are typically also something you want to include in your builds. For example, the Unit Tests and the ARM validation[6] for your infrastructure code.

[3] https://xpir.it/mag4-slc3
[4] https://xpir.it/mag4-slc4
[5] https://xpir.it/mag4-slc5
[6] https://xpir.it/mag4-slc6 and https://xpir.it/mag4-slc7
[7] https://xpir.it/mag4-slc8

Test

The hardest stage of the application lifecycle (as mentioned before) is the Test stage. In a continuous delivery workflow, continuously testing and getting feedback about the steps you executed is critical, throughout the entire lifecycle of your application. Ideally, the effort you spend on creating different kinds of tests should be distributed according to the Test Pyramid[7].

When you create a web service or an API in the traditional way, you create unit tests to test the inner workings of your methods (unit level) and API tests to validate and ensure the correct implementation of the interface (service level). At a later stage, integration tests are required to test relationships to other components or services (UI level). In the end you will need some load tests to validate whether your service can perform under load as well.

All these test types are also relevant in a serverless context. However, when you create a serverless application, the question is whether all the tests should bear the same weight as when you use a traditional development method for your application.

At the unit level, you will want to use some form of manual testing on your functions. The Azure Functions documentation describes manual testing with some http clients like Postman or Curl . As mentioned before, this is simple and easy to use, but not sustainable when you have a large number of functions that need to be tested and maintained. For automated unit testing you will want to use the capabilities of your IDE, as described in the previous chapter.

For API and integration testing (at the service and UI level) your function will need to be running and be accessible from the outside. Of course there is the emulator to run your Azure Function locally, but at some point you will want to run your function more "real".
This involves deploying and running

your function somewhere, as described in the Release phase.
When it comes to load testing, it is fairly easy to run these tests in the normal manner. But the question that we really should ask ourselves is: Is this still required? A serverless application is defined by the fact that it is small and that you only focus on the functionality and not the underlying platform.
Does it make sense to load test your function? What are you really testing – the functionality or the underlying platform?
In some cases load testing does make sense, for example if you want to test whether concurrent calls have any impact on shared data or underlying data stores. However, scalability and availability of the application should be a given when using serverless.

Release

Whereas the build pipeline is probably not very different for a serverless application, the release pipeline certainly is! There are two considerations to bear in mind when thinking about deploying your application towards production.

- How do you deal with different environments for testing
- How do you deal with different versions of the same application

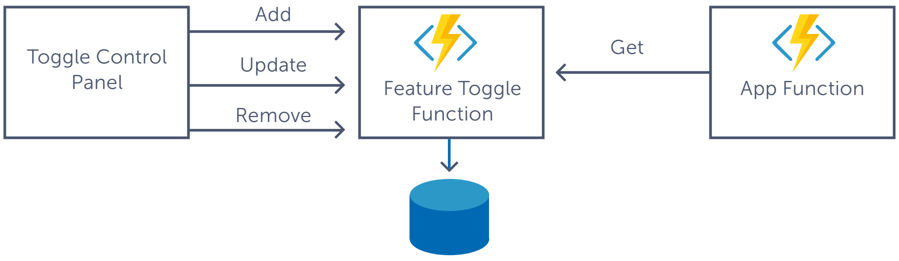
Different environments or just a different version? The first thing we need to decide when answering this question is whether or not we really need a separate environment to test our serverless application. What if we test "in production" using a new version of a function and not yet release it to the public? When it comes to the serverless application itself, i.e. the Azure Function, it does not really matter where you store it. It is the trigger or backend that introduces the need for a different environment.
There are two options for serving different functionality in your function: using feature toggles or deploying to a separate environment.

The idea of using a serverless platform is that you don't need to worry about availability, because the platform takes care of this.

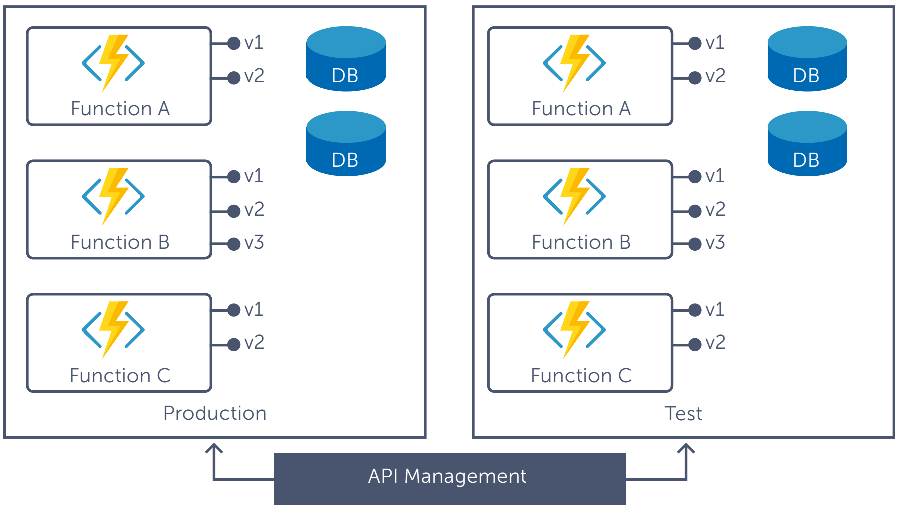
René van Osnabrugge
Microsoft Visual Studio and Development Technologies MVP

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

Using feature toggles Feature toggles are a way to modify system behavior in a running application[9]. When you use feature toggles, you need a mechanism to switch the toggle. A possible way to do this would be to create some Feature Toggle Functions that can add, update or get a value for a specific toggle from a data store. Your App Function (which implements the actual functionality) can use the Feature Toggle Functions to get the value for specific toggles, and determine its own behavior accordingly. This lets you update (or change) the functionality of your App Function at runtime.



Deploying to a separate environment If you prefer your functions to always exhibit the same behavior, a better approach might be to switch towards a different deployment for every "environment" and version, and use an API Manager to direct the traffic to the right endpoints. If you consider the fact that each new version is a new deployment, running on a different URL, you can probably imagine this will be hard to maintain and communicate with the consumers of your serverless app. An API manager like Azure API Management[10] can help you in routing your traffic to the right endpoints. The lightweight version of this is Azure Function Proxies[11].



application.organization.com/functionA?v2&apikey=test
application.organization.com/functionA?v2&apikey=prod

Routing is the new versioning.

If a consumer asks for version 2 of a service, API management will redirect him correctly to the right function implementation, without the user knowing the original URL. If you expand this concept to different environments (such as a test environment), you can configure API Management to redirect the user to a specific environment based on a specific property of the request (such as a header or request parameter, like an API key). This is completely transparent for end users and easy to configure. You can even update API Management through its own API, so that you can automatically expose new functions or versions from your release pipeline. Of course you need to take care of access control on API Management and make sure the right environments are called, but the concept is clear and transparent. The release pipeline still deploys the bits to different environments, and API Management is just another artifact that can be configured from within the pipelines.

[9] https://xpir.it/mag4-slc10
[10] https://xpir.it/mag4-slc11
[11] https://xpir.it/mag4-slc12

Creating a serverless application is not that different from creating any other application.

Kees Verhaar
Visual Studio ALM Ranger

Kees works as a consultant in the domain of Application Lifecycle Management at Xpirit. He assists organizations in transforming ideas into working code in production. Kees is passionate about exploring new technologies and opportunities whenever he gets the chance. Through his work as a Visual Studio ALM Ranger, he contributes his knowledge and real-life working experience to the developer community.

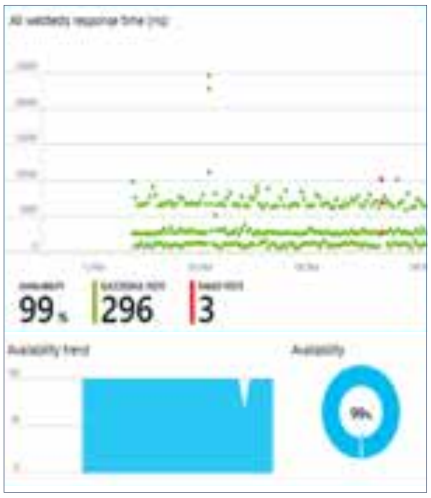
Visual Studio
ALM Ranger

Operate and Monitor

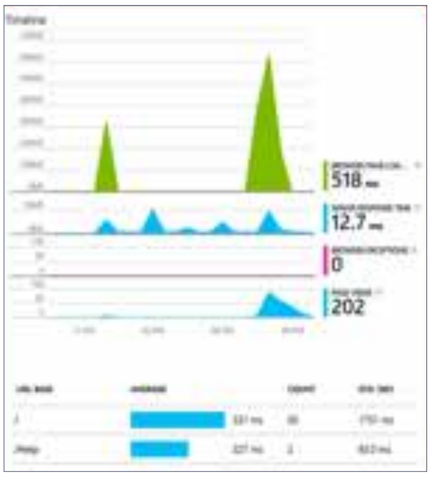
The last phase in our Lifecycle is Operate and Monitor. Monitoring is required, just like with any other application. While traditional operational monitoring on metrics such as CPU, memory and disk usage is not needed, you still have some responsibilities when you release a serverless function to the world. At least you need to make sure the function is still running as intended, and is not generating any errors. It is also very valuable to check whether the function is actually used. Azure Functions provides a number of basic monitoring features, which can be accessed from the Azure Portal or from the Azure command line.

If you need more advanced monitoring, you can implement Microsoft Application Insights, which distinguishes between four kinds of monitoring.

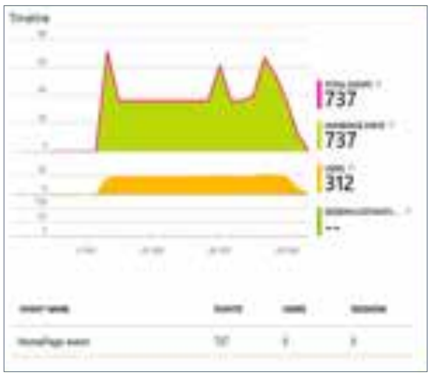
Availability



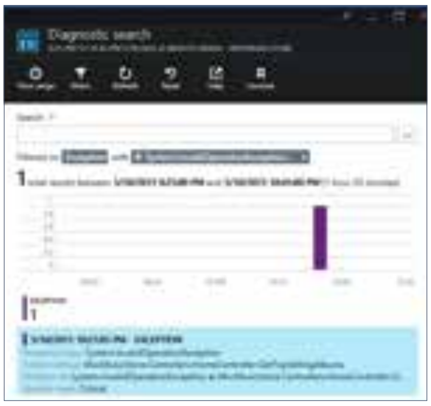
Performance



Usage



Diagnostics



The idea of using a serverless platform is that you don't need to worry about availability, because the platform takes care of this. In that respect, performance is also taken care of by the platform, except for specific situations. In addition, Diagnostics and Usage are both very relevant. You want to know whether your function is working properly, and if not, the cause of the failure. Usage statistics are valuable because you can use these metrics to decide whether you can phase out a function, or which function is a candidate for optimization.

Conclusion

Creating a serverless application is not that different from creating any other application. The tools allow you, and sometimes even encourage you, to do it in a simple way. However, usually it is advisable to think ahead and to make some decisions up front. When you consider putting your application in production, treat your serverless application just like any production application. Utilize powerful tools like an IDE, think about your test strategies, create build and release pipelines and gather metrics. Azure Functions offers great support for multiple languages and any language is the right choice. Managing your application lifecycle correctly is what makes the difference! </>

Building cloud-native Xamarin mobile apps

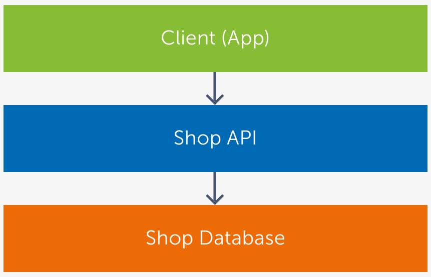
I've built numerous apps in my life and the one thing they had in common was that they all had to retrieve their data from somewhere – typically, a REST-based API. Cloud technology made it easier to implement these APIs, but our mobile software architecture hasn't changed that much. Is cloud-native technology going to change this? In this article, I'll cover several options to implement a cloud-native mobile backend using Azure and I'll explain what impact this will have on your Xamarin mobile apps.

Author Geert van der Cruijsen

Cloud-native Mobile Architecture

The cloud is already a common platform to host your mobile app backend. However, the architecture of these backends is often just the same as if they were hosted anywhere else. This doesn't have to be a bad thing since this gives you the flexibility of being able to host them anywhere. In this article, we'll look at using cloud-native features of Azure that can help you reduce your development effort and therefore enable you to adapt to your customer demands faster, and build innovative solutions.

If you've ever built a mobile backend, you've probably built something like the following example: a set of APIs or web services that expose all your business logic to the client through REST or procedural calls. If you are a .NET developer, you've probably implemented this using ASP.NET Web API. These APIs are your central access point to all data and logic that is stored somewhere in the backend. As an app developer you don't care what happens behind the API, because you just communicate with this API and its contract.



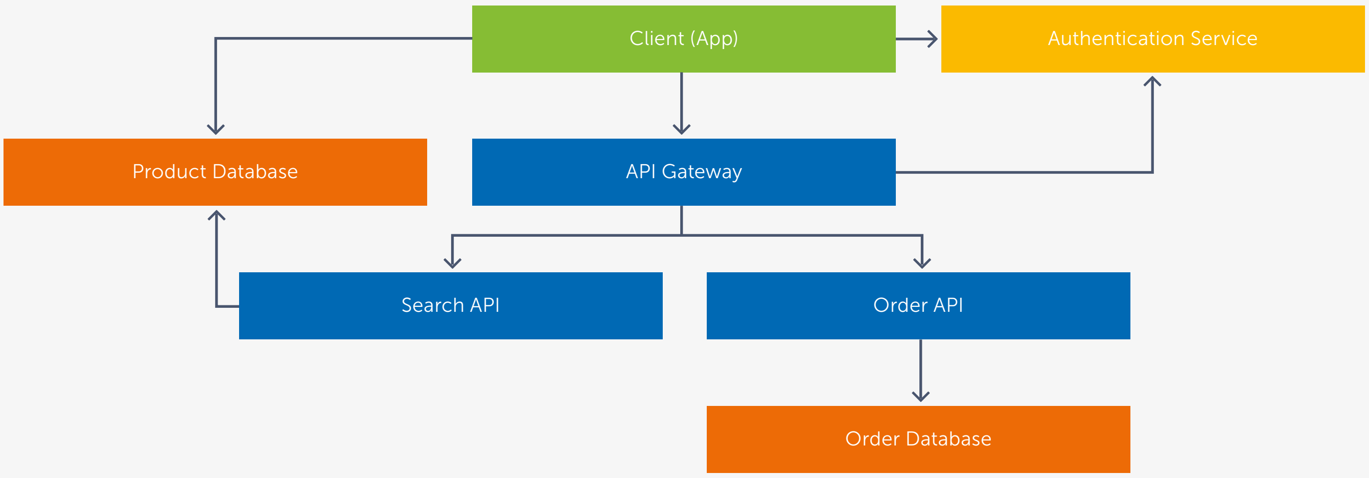
When building a cloud-native mobile backend this doesn't have to be different, but there are several options that will give you these kinds of APIs for free, so you can spend time on things that really matter instead of building all this plumbing. As an app developer, you do have to know a bit more about what happens in the backend because you'll be talking to certain components directly through different channels.

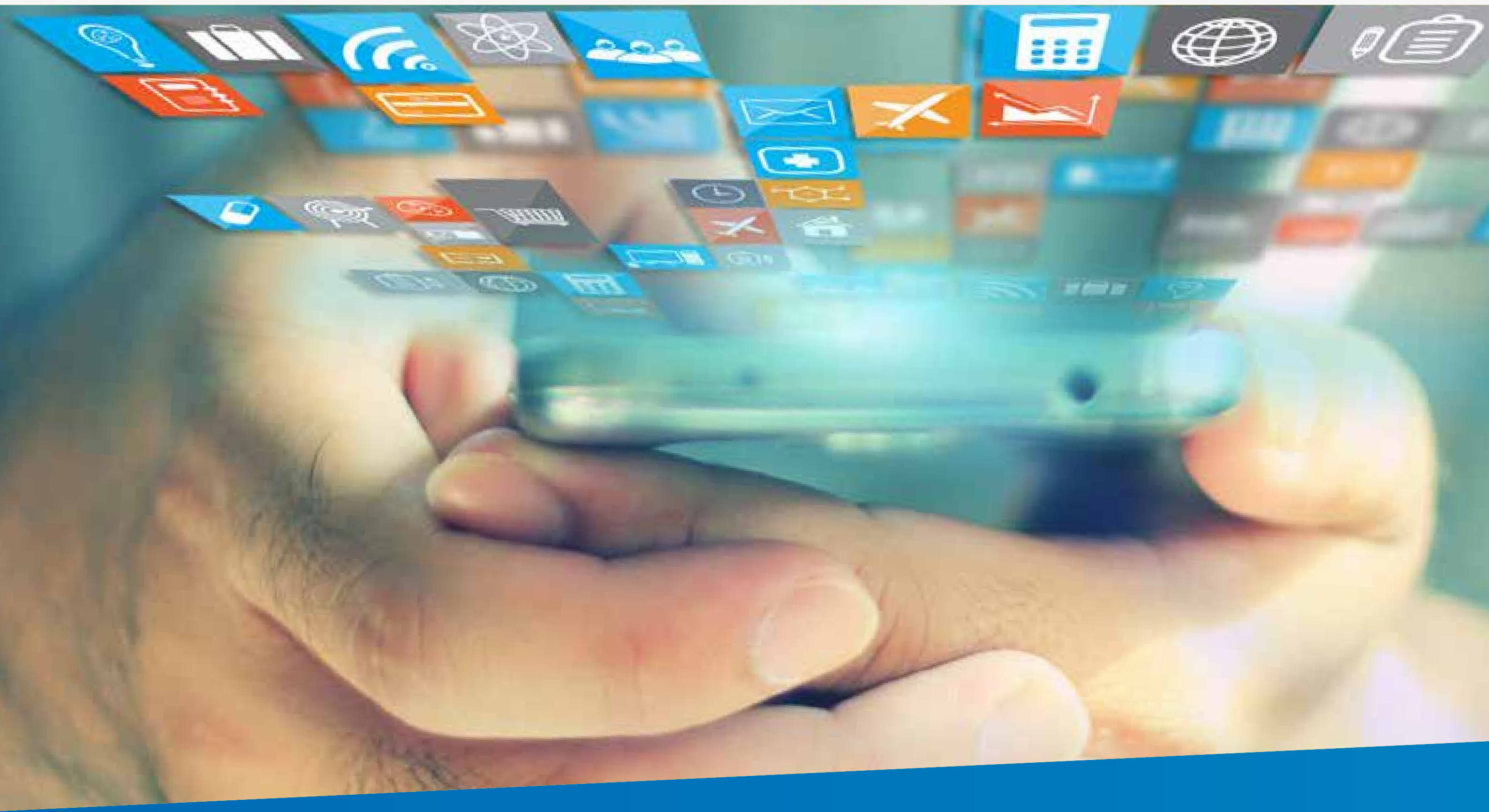
Let's look at a simple example: we want to store our products in a DocumentDB on Azure. We can directly query the DocumentDB from our client, but inserts will probably still go through some form of custom-built APIs because you want to do validations. Or there is some other business logic involved in creating a document that you don't want to have executed by your mobile app. In this simple example, we can use cloud-native components that make it super easy to build this, but the client has to know that there are two different APIs to be used which are in different locations instead of doing a GET or PUT on the same resource like they did in the past.

In the rest of this article I'll zoom in on three different cloud-native solutions that could make your life as a mobile backend developer a lot easier: DocumentDB, Azure Mobile Apps and Azure Functions.

DocumentDB

Storing data in your backend is key to most applications. DocumentDB is an Azure cloud-native NoSQL database with automatic scaling features and possibilities to replicate your data globally without making any changes to your app. It's possible to connect your app directly to DocumentDB in the cloud without creating any service layer on top of it.





```
var query = Client.CreateDocumentQuery<PointOfInterest>
        (coll nk,
        new FeedOptions{ MaxItemCount = -1,
                  PartitionKey = new PartitionKey (this.UserId)})
        .Where (poi => poi.Location.Distance(currentLocation) < 500)
        .AsDocumentQuery ();

var poiList = new List<PointOfInterest> ();
Items = await query.ExecuteNextAsync<PointOfInterest> ();
```

Benefits of a DocumentDB as mobile backend DocumentDB is a schemaless JSON database, which means that you don't have to design your data model or set up indexes up front. DocumentDB will be able to execute rich queries on top or your JSON objects that may have different properties without throwing any exceptions. This allows you to iterate and release frequently without having to do a full database schema upgrade, which can be a real pain if you want to do it often.

With DocumentDB being a cloud-native solution it's set up with cloud scale in mind. The scale supports up to millions of requests per second and has native features to replicate the data over multiple regions within Azure.

Some features that can be especially useful for mobile apps are the geo-spatial queries to query things in your neighborhood, or the use of binary attachments.

When not to use DocumentDB? While DocumentDB is flexible and offers great performance, it is not a relational database. If your data model works best as a relational model, don't try to fit it into a NoSQL database like DocumentDB. Its design just isn't made for it and you will probably kill the performance by trying to fit your relational model into the DocumentDB.

Adding a DocumentDB to your app
Adding a DocumentDB to your app is simple. There is a native SDK you can add to your portable class library through a nuget package "Microsoft. Azure.DocumentDB.Core". After adding this nuget package we can add a new DocumentClient that passes the URL to our DocumentDB, as well as a resource token to define permissions.

Now that we've set up a connection, it is possible to add items or to query your collections in the DocumentDB. To do a simple query, you can create a DocumentQuery querying on a specific Type, and create a query using Linq in the same way as you would expect any other Linq Query.

Querying on geospatial properties is part of the SDK as you can see in the sample above. On properties that are Points you can query their distance without having to use any special code in your Linq queries.

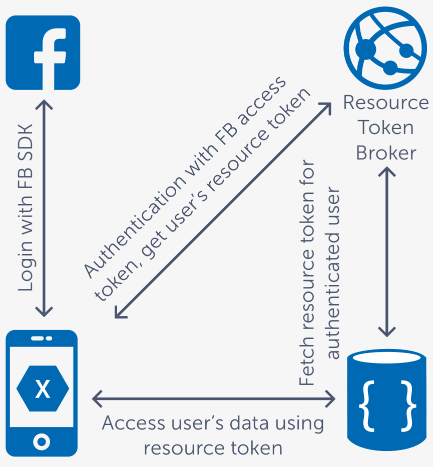
Inserting items is just as simple. You just need the DocumentClient and a link to your collection in the DocumentDB. With those properties you can insert any C# object, as long as the object is serializable as JSON. By adding the UserId to the object, the permissions are automatically set up correctly so that only our user or other users who have been given permission can find this item.

```
pointOfInterest.UserId = this.UserId;
var result = await Client.CreateDocumentAsync
(collectionLink, pointOfInterest);
pointOfInterest.Id = result.Resource.Id;
```

DocumentDB does not have any authentication mechanism built in and will just require the user-specific resource token you pass to the client constructor. Retrieving this token can be done by adding a separate Azure App Service app that will do the authentication for us.

To set this up you have to follow the following steps.

1. Create a DocumentDB collection where you define a partition key on the UserID property.
2. Log into your app using an OAuth mechanism.
3. Use the OAuth token to authenticate to the resource token provider API that is part of Azure App service.
4. Resource token API will request a DocumentDB resource token with read/write permissions on the collection.
5. Resource token API returns the resource token to the app.
6. App passes the resource token to its queries.


Resource Token Broker
Login with FB SDK
Authentication with FB access token, get user's resource token
Fetch resource token for authenticated user
Access user's data using resource token
X
{ }

Azure Mobile Apps

The most well-known solution to build your cloud-native mobile backend in Azure is to use Azure Mobile Apps. Azure Mobile Apps are part of Azure App Service, Microsoft's solution for building a PaaS (Platform as a Service) backend solution. Azure Mobile Apps offer specific mobile features on top of Azure App Service.

Benefits of an Azure Mobile Apps backend Azure Mobile Apps offer specific mobile features such as Client SDKs for iOS, Android, Windows, Xamarin and even Apache Cordova.

There is out-of-the-box support for OAuth 2.0 with most social networks to make authenticating easy and there is integration with Azure Notification Hubs to send push notifications to all platforms using a single backend.

For proof of concepts or relatively simple apps, Azure Mobile Apps includes Easy Tables. This is a feature that uses an SQL Azure database behind the scenes. What makes Easy Tables so easy is that the columns and tables will automatically be generated when data is added to the database.

The most powerful feature of Azure Mobile Apps is the possibility to create offline data sync between your backend and your mobile app. To enable the offline data sync you'll have to implement a local storage using SQLite or SQLCipher, and set up the out-of-the-box sync with the data provider that is part of the Azure Mobile App. This feature allows you to create the offline data sync with only a couple of lines of code, and without having to think of conflict resolution of sync errors due to network issues.

Offline synchronization of your data offers a lot of new business scenarios in your apps because initially your UI will save all changes locally in your SQLite database and sync this data to the backend asynchronously in the background. This can make performance of the app look great because you don't have to wait because of the communication delay to your backend, which can be slow depending on the location and reception your phone has when using the app. Imagine building such a synchronization mechanism yourself. It would be quite hard to build, especially if it includes conflict resolution and sync errors fixing.

When not to use Azure Mobile Apps
Azure Mobile Apps can be used for all kinds of purposes, from small apps to enterprise apps. Adding custom ASP.NET Web APIs within Azure Mobile Apps can handle all kinds of scenarios. Easy Tables works great for small apps or POCs, but is limited when building larger apps with complex data models. It is wise to use custom APIs within the Azure Mobile App when building more complex mobile backends so you have full control over your business logic on the server side.

Using Azure Mobile Apps Offline synchronization for a table within Azure Mobile Apps to your native mobile app can be created in under 50 lines of code. To add an Azure Mobile App to your app, first create a new Azure mobile app in the Azure Portal. After that you can start coding your app by adding a nuget package called "Microsoft.Azure.Mobile.Client. SQLiteStore". This will give you access to the SDK to sync data from a local SQLite database to an SQL Azure database.

After adding the nuget package, the SDK has to be initialized by adding the Init method in the FinishedLaunching method of the AppDelegate class in iOS.
On Android you don't have to initialize the SQLitePCL; you only need to add the first line within the OnCreate Method of your MainActivity.

```
Microsoft.WindowsAzure.MobileServices.CurrentPlatform.Init();
SQLitePCL.CurrentPlatform.Init();
```

After initializing the SDK you have to initialize the sync of a table, in this case a table containing Friend objects which is just a plain old C# object. Create a new class called AzureMobileService, and this is where you add a method Initialize. In this method, you'll first have to create a new MobileServiceClient and pass the url of the Azure Mobile app you created. On the client, you can define a new table of type Friend.

```
public async Task Initialize()
{
    if (Client?.SyncContext?.IsInitialized ?? false)
        return;

    var appUrl = "https://friendsappdemo.azurewebsites.net";
    var path = "friends.db";
    Client = new MobileServiceClient(appUrl);
    path = Path.Combine(MobileServiceClient.DefaultDatabasePath, path);

    var store = new MobileServiceSQLiteStore(path);
    store.DefineTable<Friend>();

    await Client.SyncContext.InitializeAsync(store);
    friendTable= Client.GetSyncTable<Friend>();
}
```

After initializing, you need to implement three methods, a sync method that will synchronize the changes between your local database and the SQL Azure database and two other methods to add Friends and to retrieve Friends.

```
public async Task SyncFriends()
{
  await friendsTable.PullAsync("allFriends", friendsTable.CreateQuery());
  await Client.SyncContext.PushAsync();
}

public async Task<IEnumerable<Friend>> GetFriends()
{
  await Initialize();
  await SyncFriends();

  return await friendsTable.OrderBy(c => c.DateUtc).ToEnumerableAsync(); ;
}

public async Task<Friend> AddFriend(Friend friend)
{
  await Initialize();
  await friendsTable.InsertAsync(friend);

  await SyncFriends();
  return friend;
}
```

This is all the code you need to set up offline sync. On my GitHub page you can find a fully working sample that runs on all mobile platforms.

Implementing an Azure Function should focus on a specific task that isn't covered by the functionality of DocumentDB APIs or Azure Mobile App Table Storage in your Mobile backend.

Azure Functions

We've looked at two different ways of implementing a cloud-native, backend solution in Azure. With the upcoming trend of serverless architectures, Azure Functions could also play a role in acting as a mobile backend. Azure Functions deliver nano services that really take advantage of the cloud with only paying for what you use and can be scaled up and down quickly.

The functions are small pieces of code that can be triggered by HTTP calls, a timer schedule, or by new items in blobs or queues. There are several business scenarios you could image where this would work for mobile apps. Azure Functions also have a set of output bindings to store output on several different data storage types in Azure. Some of these are the DocumentDB and Azure Table storage as part of Azure Mobile Apps. Another output type consists of Mobile push notifications using Azure Notification Hub.

This magazine contains an article by my colleague Pascal Naber explaining all the details of creating an Azure Function, so we won't go into the specifics of implementing a Function.

Implementing an Azure Function should focus on a specific task that isn't covered by the functionality of DocumentDB APIs or Azure Mobile App Table Storage in your Mobile backend.

This way you can compose your perfect mobile backend out of cloud-native components in the same way as I described this in the first chapter.

Conclusion

There are multiple ways of building your mobile backend in Azure. Hopefully this article has made you think about the options you can use instead of just lifting and shifting a mobile backend from your on-premises solution.

In the future, we will see more and more mobile backends use the best tools for the job that could, for example, be a single backend composed of several types of storage, out-of-the-box APIs, or custom built APIs.

This is why it is important to have knowledge of the various techniques you can use. The best part of these cloud-native solutions is that they are easy to create and scale, but also to tear down. This makes them perfect for proof of concepts or small apps where you have the possibility to scale up or down, depending on the success of your application.

If you want to get more technical details on implementing these Azure features, check out my GitHub Page on github.com/geertvdc/cloudnative-appdemo to find a full working sample with DocumentDB and an Azure Mobile App. </>

Geert van der Cruijsen
Microsoft Visual Studio and Development Technologies MVP

Geert is a mobile software architect with years of experience in building cross-platform applications using Xamarin & .NET. As a Lead Consultant at Xpirit, Geert helps customers define and improve their mobile strategy, vision and technical implementation regarding mobile application development and ALM. Geert is also a Xamarin Certified Trainer, delivering training to mobile developers. Geert is an active member of the Xamarin and Windows UWP community, has spoken at several conferences, and is co-organizer of the Dutch Mobile .NET Developer's meetup.

The Ultimate Education Destination

2017 Orlando

ROYAL PACIFIC RESORT AT UNIVERSAL ORLANDO
NOVEMBER 12-17

LIVE! 360
TECH EVENTS WITH PERSPECTIVE

Live! 360: A Unique Conference for the IT and Developer Community

- 6 FULL Days of Training
- 5 Co-Located Conferences
- 1 Low Price
- Create Your Own Agenda from Hundreds of Sessions
- Expert Education and Training
- Knowledge Share and Networking

CONNECT WITH LIVE! 360

twitter.com/live360
@live360

facebook.com
Search "Live 360"

linkedin.com
Join the "Live! 360" group!

EVENT PARTNERS
Microsoft Magenic

PLATINUM SPONSOR
SMARTBEAR

SUPPORTED BY
Visual Studio msdn magazine Redmond Channel Partner

LIVE! 360

TECH EVENTS WITH PERSPECTIVE

NEW: HANDS-ON LABS

Join us for full-day, pre-conference hands-on labs Sunday, November 12.

Only $595 through August 11

REGISTER NOW

REGISTER BY AUGUST 11 AND SAVE $500!*

Use promo code L360MAY2

*Savings based on 5-day packages only. See website for details.

Check out our other 2017 events for Developers and IT Pros:

- Visual Studio Live! - vslive.com
- TechMentor - techmentorevents.com

5 GREAT CONFERENCES, 1 GREAT PRICE

Visual Studio LIVE! EXPERT SOLUTIONS FOR .NET DEVELOPERS

Visual Studio Live!: Code in Paradise at VSLive!™, featuring unbiased and practical development training on the Microsoft Platform.

SQL Server LIVE! TRAINING FOR DBAs AND IT PROS

SQL Server Live! will leave you with the skills needed to drive your data to succeed, whether you are a DBA, developer, IT Pro, or Analyst.

TECHMENTOR IN-DEPTH TRAINING FOR IT PROS

TechMentor: This is where IT training meets sunshine, with zero marketing speak on topics you need training on now, and solid coverage on what's around the corner.

Office & SharePoint LIVE! ON-PREMISE, CLOUD & CROSS-PLATFORM TRAINING

Office & SharePoint Live!: Today, organizations expect people to work from anywhere at any time. Office & SharePoint Live! provides leading-edge knowledge and training to work through your most pressing projects.

Modern Apps LIVE! MOBILE, CROSS-DEVICE & CLOUD DEVELOPMENT

Modern Apps Live!: Presented in partnership with Magenic, this unique conference delivers how to architect, design and build a complete Modern App.

PRODUCED BY

Redmond Magazine · Virtualization & Cloud Review · Visual Studio Magazine · 1105 MEDIA

LIVE360EVENTS.COM

Authentication as a Service with Auth0

Authentication is an important factor in the success of applications and their architectures. Serverless architectures are no exception to this. Authentication is an area in which serverless computing is becoming the new norm. Without any doubt, authentication is one of the most complex features to implement correctly. When you use a serverless platform for authentication, you can focus on actual business functionality of the application instead of hosting and running an authentication platform. One of the leading serverless Authentication as a Service (AaaS) platforms is Auth0. Others are Azure AD (B2C) and Okta.

Author Chris van Sluijsveld

Auth0

Auth0 is an enterprise-grade platform for modern identity. It helps you build authentication for your applications using a frictionless platform.
Auth0 offers an easy-to-use dashboard that allows you to start adding authentication to your applications. From simple logins using social accounts (Microsoft, Google and Facebook) to enterprise logins using Azure Active Directory, for example. Auth0 provides you with total control over the functionality without having to manage and maintain the platform.
Auth0's Webtask.io offers a serverless platform to add customizations and other features to the authentication flows.
In this article, I will explain how easy it is to build WebApps and APIs with Authorization with Auth0. In this example, we will use an ASP.NET Core WebApp, Azure API management, and ASP.NET Core WebAPI, all connected to Auth0.

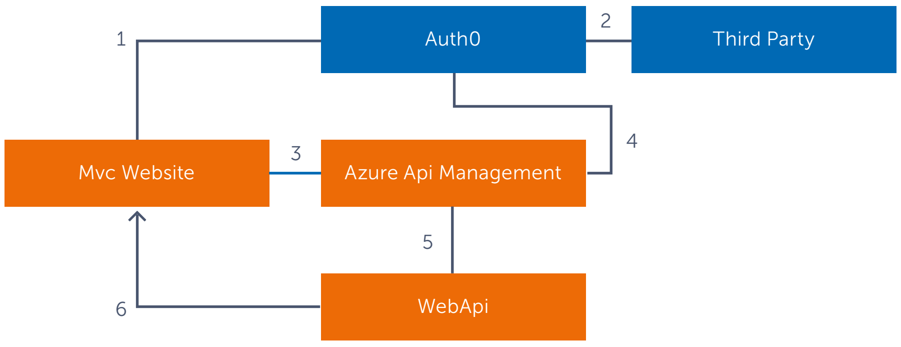


Figure 1 shows the architecture for this example.

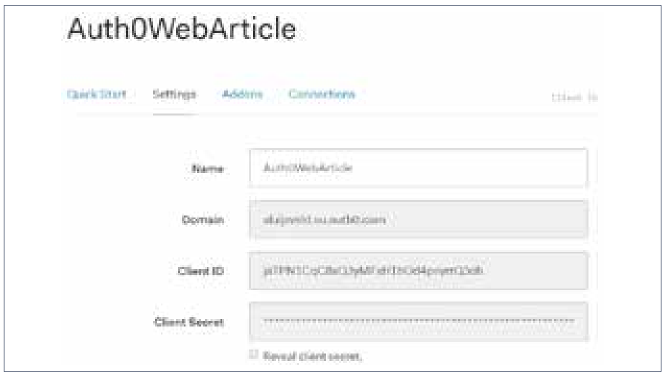
The application flow looks like this.

1. User visits MVC Website.
2. User logs in on Website and is redirect to Auth0.
3. Uses logs in using a third party (Google or Microsoft) login provider. User gets sign-in metadata and a token for API usage.
4. Website calls WebAPI using the token retrieved during login.
5. Azure API Management validates signing key (RS256) using config endpoint.
6. Azure API Management forwards the request to WebAPI.
7. WebAPI returns claims information to Website in JSON message.

First of all we start by creating an application in the Auth0 dashboard at https://manage.auth0.com.
After registration, you can get started without any costs. With up to 7000 active users in the free tier, this should be enough to get started. After logging in you start with creating a new client.

We get a Domain, Secret and client ID that we need in our WebSite, API Management platform, API to configure our authentication code.

Make sure that you specify the "Allowed Callback URLs" and "Allowed Logout URLs" correctly for either development (local machine) or hosting platform (for example Azure). Switch "JsonWebToken Signature Algorithm" to RS256 to make the ASP.NET Core OpenID middleware work correctly.



Name: Auth0WebArticle
Domain: [illegible]
Client ID: [illegible]
Client Secret: ••
☐ Reveal client secret.

Website

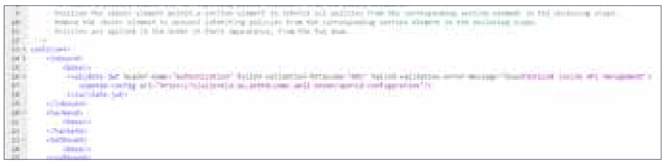
Configure the OpenIdConnect middleware using the clientId, clientSecret, and Authority information from Auth0. ASP.NET Core offers standard middleware for this purpose. It is located in the "Microsoft.AspNetCore.Authentication.OpenIdConnect" package. When you add this package to your project, you have to configure the middleware of your app. This can be done by adding the following code:

```
app.UseIdentity();
app.UseOpenIdConnectAuthentication(new OpenIdConnectOptions
    {
        ClientId = Configuration["ClientId"],
        ClientSecret = Configuration["ClientSecret"],
        Authority = Configuration["Authority"],
        ResponseType = OpenIdConnectResponseType.Code,
        GetClaimsFromUserInfoEndpoint = true
});
```

Azure API Management

Azure API management is the first barrier for successfully calling the exposed APIs. The Azure API Management platform is also the first place where you can check and validate the JWT (JSON Web Token) tokens for authenticated access to the APIs. Azure API Management is a Platform as a Service (PaaS) solution inside Azure. API Management helps organizations publish APIs to developers inside and outside of your organization. It provides the core competencies to ensure a successful API program through developer engagement, business insights, analytics, security, and protection.

After adding the WebAPI to Azure API management it is possible to specify policies for each individual operation or for all operations at once. In this case, we will add an inbound policy to all operations using the code-view. One could see these polices as another form of serverless computing, where you can easily customize the API management platform without worrying about the hosting or deployment of those customizations. It just runs in the platform provided.
The following segment of xml has to be added to the inbound policies for your API. In this case it checks for a valid JWT that has not yet expired, and uses the OpenID configuration endpoint of Auth0 to check the signing key of the incoming JTW token.



```
[illegible]
```

When you try the API in the Azure API Management developer portal, you should see the message "Unauthorized inside API Management" if no id_token is passed to the ASP.NET Core WebAPI inside the authorization header of the request.

WebAPI

The ASP.NET Core WebAPI is the last part that wants to check the incoming token to authenticate the user. In the same way as with the ASP.NET Core WebApp, there is a standard library to authenticate the use of JWT tokens. This can be done by adding the following lines of code to your startup middleware.

All that is left to do now is marking your controller operations with the [Authenticate] attribute. When you test the application, you should be able to log in using a social media account, see your claim information, and call the API to validate the id_token.

```
var options = new JwtBearerOptions
{
Audience = Configuration["JWTOptions:clientId"],
Authority = $"https://{Configuration["JWTOptions:domain"]}/"
};
app.UseJwtBearerAuthentication(options);
```

Without knowing the nitty gritty details of Auth 2.0 or OpenIdConnect you can still easily integrate it into your solution.

Conclusion

Auth0 as an Authentication as a Service offers a really powerful platform. It is easy to integrate in your solution with a simple registration and using the values provided. I believe this is the enormous power of a serverless platform. Without knowing the nitty gritty details of Auth 2.0 or OpenIdConnect you can still easily integrate it into your solution. You just provide your code and it just works. You can also change functionality on the fly without needing any redeployment of your code. Meanwhile, the platform does all the work and is totally managed for you. The code for the sample in this article is available on https://xpir.it/mag4-auth. </>

Chris van Sluijsveld

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

Backup and Restore in Azure Service Fabric

Creating and restoring backups of stateful services can be challenging. In this article you will learn how it works and what you can do to make this process much easier.

Author Loek Duys

What is Azure Service Fabric?

Before we start creating backups, I will first briefly introduce Service Fabric and its application model. Most companies have many applications to run, usually on multiple, over-dimensioned servers, sized on peak loads. This means that most of the time, server resources are not utilized efficiently. Service Fabric creates a virtual pool of computing resources by joining multiple servers – or nodes – together into a cluster. Service Fabric then adds mechanisms to optimize the use of the underlying cluster resources. It automatically takes care of application placement and upgrades, cluster health monitoring and rebalancing applications, based on their resource consumption.

Application Model

Service Fabric applications consist of one or more services that work together to automate business processes. A service is an executable that runs independently of other services, and is composed of code, configuration and data. Each element is separately versioned and deployable. In this model, 'code' means the service binaries. Xml files which hold 'configuration' have custom service settings, such as connection strings and security settings. Finally, 'data' is any static data your service uses, e.g. pictures and script files.

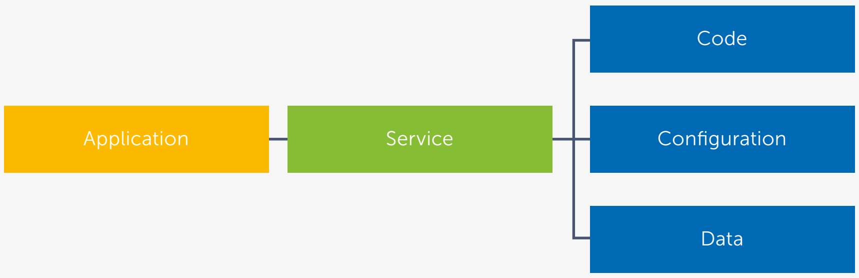


Figure 1 Application Model

Creating an application instance requires an Application Type; this is the template that specifies which service instances should be created as part of the application. This concept is similar to object-oriented programming. The Application Type is like a class definition, and the application is a named instance. Multiple application instances can be created from one Application Type.

The same concept applies to services. A Service Type defines the code, data and configuration for the service, as well as communication endpoints used by the service for interaction. Multiple named service instances can be created using one Service Type. An application specifies how many instances of which Service Types should be created.
Both Application Type and Service Type are described through XML files. Every element of the application model is versioned and deployed independently.

System Services Service Fabric itself runs as services on the cluster. These system services manage the cluster and are used to deploy and monitor the services you run on the cluster yourself. One of the system services is the Fault Analysis Service. This service plays a role in restoring backups. You'll learn more about this service later.

How Stateful Services work

Stateful services keep their data close by, stored in memory and on a local disk. To enable large scale projects with many concurrent users, stateful services can be distributed across multiple nodes. Each instance of a stateful service is called a replica. Each replica stores its own chunk of the total service state. This means that your data is divided across multiple service replicas.

All Replicas are made highly available through an automated data replication system, which copies the state across multiple cluster nodes during transactions. So if one cluster node fails, your data will be safe and your service continues to be available. It also means that you may need to query multiple Replicas to get all your data.

And finally, it means that if you want to create or restore a backup of your service, you will need to do this separately for every partition.

Creating a backup of a Replica

The state of your stateful service is safely stored across multiple nodes. However, your data can still be lost, due to the loss of your entire cluster, or to human error. To protect you from data loss it is sensible to create regular backups of your service state.

Backup types There are two types of backups:

- Full backups - A full backup contains a complete copy of the state of one replica. Full backups can become quite large as your service state grows over time.
- Incremental backups.

An incremental backup builds on top of a full backup and any later incremental backups. It contains only the changes made since the last full backup. Because of this, a partial backup is usually smaller and faster to create than a full backup.

How often you create backups depends on the acceptable amount of data loss you can afford for your service. Mission-critical data and data that constantly changes, should be backed up more often than other types of data.

Adding code to create backups
Creating backups of Service Fabric Services requires you to add some code to your stateful service. First of all, you will need to call the existing method BackupAsync and pass it an instance of BackupDescription which contains a callback. This callback – or delegate – will be executed once the backup is created to perform additional actions, if needed.
The callback delegate has the following signature:

```
Func<BackupInfo, CancellationToken,
Task<bool>> BackupCallback
```

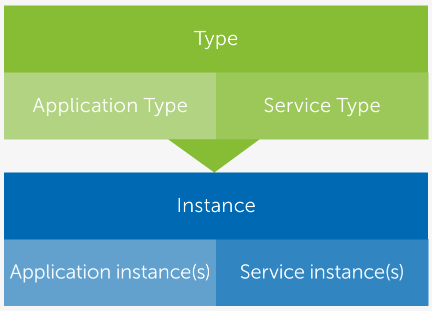


Figure 2 Types versus Instances

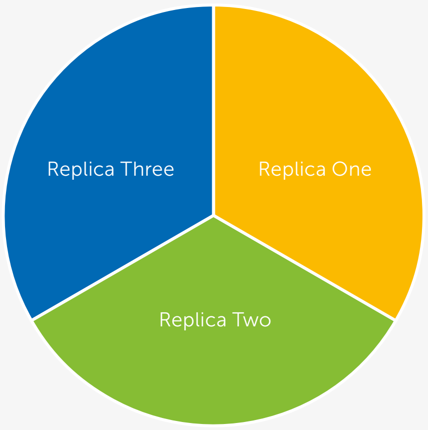


Figure 3 Replicas of a stateful service

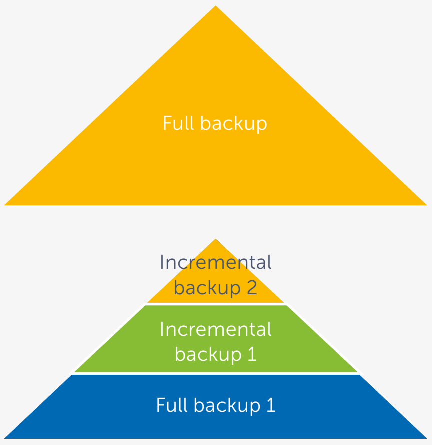


Figure 4 Types of backups

The CancellationToken can be used to cancel the operation. The returned Boolean indicates the success of the operation. The provided BackupInfo class contains useful information, for instance the backup folder, its type and the version. To keep your Backup files safe, you should copy them away from the cluster to a central store, like Azure Blob Storage. This way, you can recreate a lost cluster using the Backup files.

Remember that this code needs to be executed for each replica of your service. To make this process reliable and repeatable, you could for instance create a new Service Fabric service that periodically backs up other services in the cluster.

Restoring a backup of a Replica

Figure 5 shows the process of manually restoring a replica. The restore process is more complicated than creating a backup. It requires the help of one of the Service Fabric system services we discussed earlier; the Fault Analysis Service.

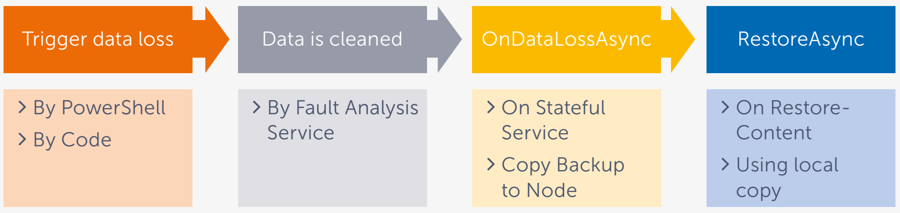


Figure 5 Restoring a Replica using triggered data loss

Triggering 'data loss' To begin the process of restoring a backup, Service Fabric must invoke the method OnDataLossAsync on your stateful service replica. This can happen in the following situations:

- The call to OnDataLossAsync can happen because Service Fabric detected data loss itself. This will happen if there is a problem with the cluster node that hosts the service. For instance, if there is a disk failure.
- It can also be triggered by using code:
  ```
  FabricClient.TestManager.StartPartitionDataLossAsync([..]);
  ```
- And finally, it can be triggered by using PowerShell
  ```
  Start-ServiceFabricPartitionDataLoss:
  ```

Cleaning existing data The last two options are triggered by the program code; the Fault Analysis Service is called from "under the hood", and it ensures that the restore operation is performed for the targeted service replica. The amount of data that will be lost by these operations depends on the value of dataLossMode. It has two options:

1. PartialDataLoss – This option indicates that only pending replications will be lost.
2. FullDataLoss – This option indicates that all data is lost.

Invoking OnDataLossAsync

After the data has been removed, OnDataLossAsync is invoked on the replica, so it can begin to restore its state. It is invoked with a parameter RestoreContext which contains an operation called RestoreAsync. Before you can call this method to restore a backup, you will need to download the backup files from your central storage back to a temporary folder on the cluster node. Then you must let Service Fabric know where to find the Backup files. For this, you can use the RestoreDescription struct. The RestoreDescription that is passed to RestoreAsync is used to inform Service Fabric where you have placed the Backup files locally. When restoring a full backup, only the one folder that contains it needs to be downloaded.

When restoring an incremental backup, all previous incremental backups and the latest full backup need to be down-loaded. (As explained in Figure 4.)

And now we wait... After doing all this, Service Fabric will restore your Replica. Depending on the size of the backup to restore, this can take quite some time. You can monitor progress by using the Service Fabric Explorer. It will start out with reporting errors, and show your stateful service with status 'In build'. This can be seen in Figure 6.

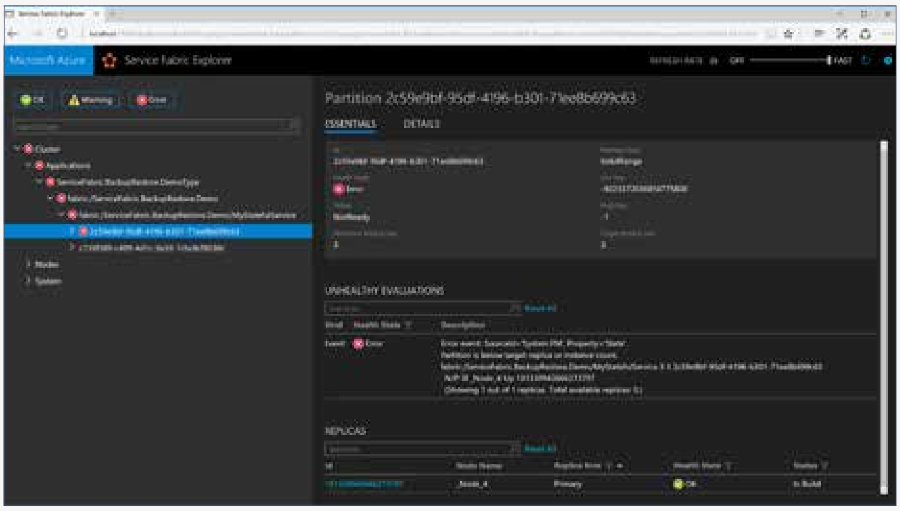


Figure 6 Service Fabric Explorer

After a while your service will become healthy again. Remember that this code needs to be executed for every replica of your service, just like the creation of the backup. It's important to note that all replicas that are not currently restoring a backup remain healthy and will continue to operate normally.

Help!

The process of creating and restoring backups and keeping them safe is quite complicated. Fortunately, there are some open source initiatives that can help you. For instance, there is an Azure Quick Start example project that shows how to use Azure Blob Storage as a central backup store in InventoryService.cs: https://xpir.it/mag4-sf1

Another option is to use my open source library "ServiceFabric.BackupRestore", available on GitHub[1] and Nuget[2]. This library provides a class called BackupRestoreService that is derived from StatefulService.

By using this class instead, you are provided with three new methods:

1. BeginCreateBackup – this method calls BackupAsync on the service and uses an injected helper class to copy the Backup files away from the cluster.
2. ListBackups – this method lists information about all Backups that were created earlier using BeginCreateBackup.
3. BeginRestoreBackup – this method is called with the information listed by ListBackups and triggers data loss, copies the Backup files from the central store to the cluster node, and invokes RestoreAsync with the proper information.

By using this library, you can create and restore backups with just a few simple lines of code. You can use the readme document and demo application to help you get started.

Loek Duys
Microsoft Azure MVP

Loek is a cloud software architect, public speaker and Microsoft Azure MVP who focuses on creating secure, scalable, and maintainable systems. He is always looking for ways to leverage the latest additions to Microsoft Azure, to help companies make the transition into the Cloud. As an active member of open source projects, he likes to share knowledge with other community members.

MVP Microsoft® Most Valuable Professional

Conclusion

In this article, you have learned how to create and restore Backups for your stateful reliable services in Azure Service Fabric. By creating backups and storing them away from the cluster, you can deal with disasters caused by full cluster failure and human errors, for instance accidental deletes. Creating and restoring backups is complicated. However, using existing code and libraries can make life simpler. </>

[1] https://xpir.it/mag4-sf2
[2] https://xpir.it/mag4-sf3

Creating and restoring backups is complicated.

Containerization in modern IT

Modern application development uses container technology more and more, whether this is for running backend or frontend services, or even for developer tooling. Many organizations such as system integrators, independent software vendors, and cloud solution specialists are adopting containerization to keep up with the demands of large scale, distribution, and modern development and operations practices. Many others are jumping on the container bandwagon because everyone seems to be doing it. Instead of blindly following along, it is good to investigate and answer a number of essential questions. What are compelling reasons to use containers or container technology, and what does it mean to use it as a foundation for your application development?

Author Alex Thissen

8 compelling reasons to use container technology:

1. Higher degree of hardware abstraction

Using containers to run your software means that you are utilizing host machines at a level where the individual machine doesn't matter anymore. Instead of caring about single machines with well-known names and maintaining and patching these, the operations team can focus on a cluster of host machines. This cluster is a fabric of the host machines, woven together by cluster software to form a contiguous set of resources for computing, memory, and storage. Each machine is built with commodity hardware and is expendable. The cluster can grow and shrink whenever new hosts are added or defective hosts are taken out of commission. The containers will be running on the accumulation of resources managed by the cluster software, without knowing or caring about the individual machines.
It is an important trend in the exploitation of cloud applications to try and pay only for the resources actually consumed. Containers enable more optimal use of the resources in the cluster. The cluster software will allocate and assign the running containers based on the requirements that the container images and application registration indicate. These requirements might include a particular operating system, memory constraints, and the required CPU resources.

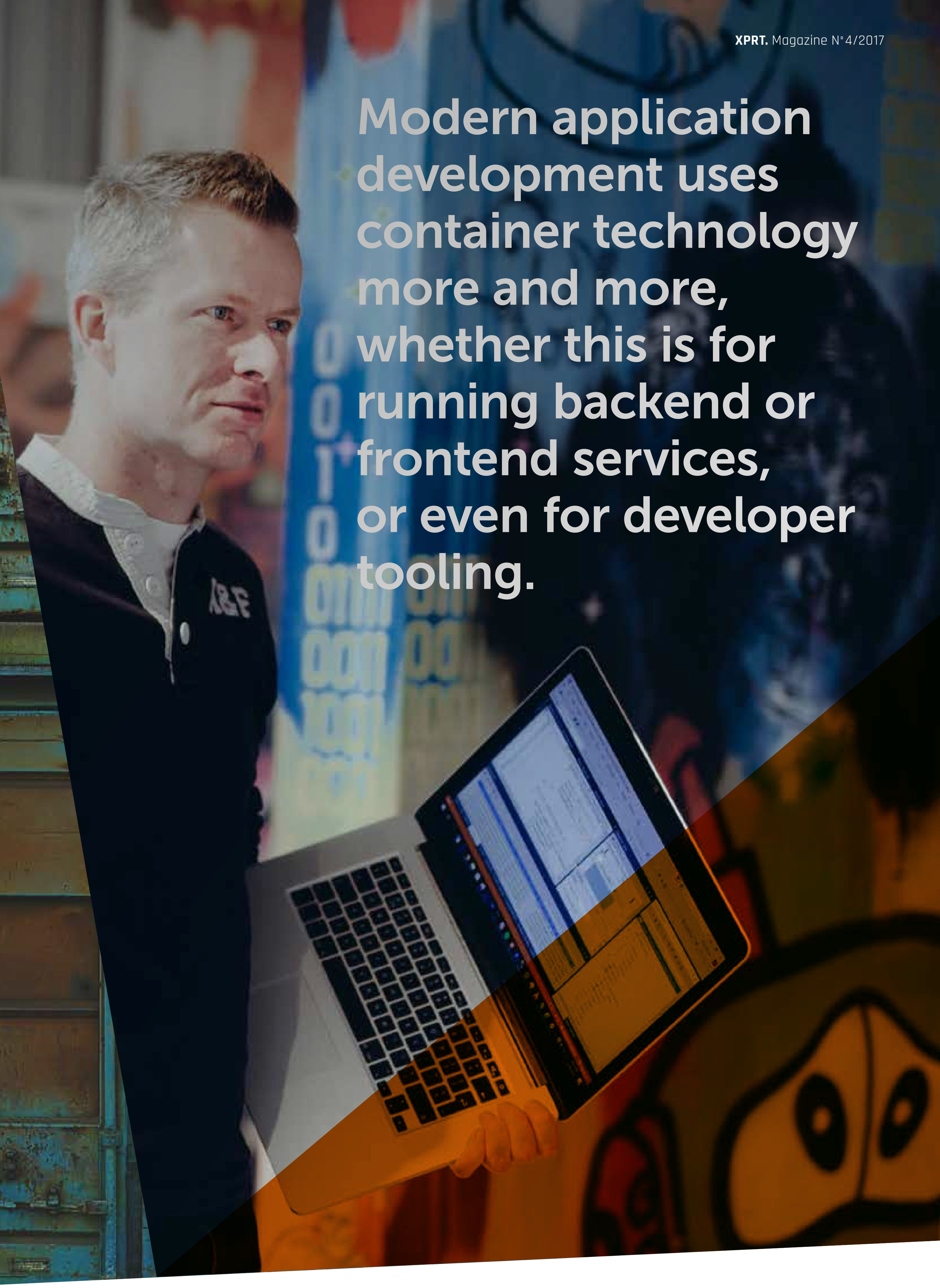
It can decide what to run and where, while taking into account the constraints that were given. The cluster might also shift containers around whenever it sees fit for a better utilization of the available resources. At the end of the day it is possible to achieve much higher density of running containers and hence applications on the cluster, when compared to running applications on dedicated machines. Fewer machines are needed to run the same workload, which results in lower costs of operations.

2. Scalable

The cluster offers a pool of resources that the containers can request for operations. Whenever the need arises for more resources and bigger scale, the cluster can spin up additional container instances from the images and allocate more resources per container, if necessary. As long as there is a surplus of resources it is easy to scale up, and down afterwards.
When resources start running out, good capacity planning will make sure more hosts are added to the cluster for provisioning containers. Smarter cluster software can manage scale better. Because containers are very fast to spin up, the scaling process itself is a matter of seconds instead of minutes or hours.

3. Freedom in hosting

Ideally, the cluster software is installed on virtual machines to easily provision a new cluster when necessary, or to add new hosts without having to care about the actual hardware. These virtual machines may reside in the cloud at any cloud provider such as Amazon AWS, Google Cloud or Microsoft Azure, but they can also be located on-premises at your own organization. There is no restriction to run the cluster anywhere specific, although in some situations there is a benefit in choosing for a



Modern application development uses container technology more and more, whether this is for running backend or frontend services, or even for developer tooling.

particular cloud provider. For example, Microsoft offers Azure Container Service which provides automatic provisioning of Kubernetes, Docker Swarm or DC/OS clusters. The various cluster software offerings run on different operating systems, both Linux-based and various Windows Server editions.

4. Container images are code and configuration
Container images combine the binaries built from code and the configuration of the machine into a single entity. This inseparable set offers better control, stability and predictability than before.
In traditional deployment scenarios, the code is deployed to target machines separate from the configuration. This led to the rise of Desired State Configuration, where script or a declarative language of some form would describe the configuration for the application code and its requirements at a host level. The code and resulting binaries were usually handled by the development team. That team also describes the required configuration in documentation.

The operations team would then get scripts from the development team to prepare the host machines and to do final configuration of the software components. This turned out to be error-prone, but also required multiple environments to separate the various stages in an application's lifecycle, such as testing, acceptance and production. Containers ensure that our applications will always run in exactly the same context over different container hosts.

5. Immutable deployment artefacts
The DevOps team builds new application components by writing and testing code, and compiling it into binaries. In a container world, they will also prepare the container images that have all of the prerequisites for the components, and deploy the new components onto them. The container image is configured to run correctly and then deployed to a central image repository. From that repository, the host can download and run the images. Once the DevOps team releases the container image to the repository it cannot be changed anymore. The images are immutable and can be guaranteed to arrive at the hosts without any changes. Any changes that occur are in the state of a running container, but never the image.
The DevOps team has full control over what will be running in the containers, and does not depend on human intervention of configuration. Moreover, it is no longer necessary to wonder whether a container image can still run correctly at a later time. Whenever a compatible cluster with the correct operating system and kernel version is available, container images are guaranteed to run, since it is a frozen set of code and configuration with no alterations since creation. Rolling back deployments and maintaining a running application suddenly becomes a lot easier.

6. Stages instead of environments
Having Development, Testing, Acceptance and Production environments (DTAP for short) might be a thing of the past when containers and clusters come into the picture. Traditionally there are separate environments to indicate the level of quality and trust in your application and its components. Now you can think of the various environment stages that an application goes through as being reflected by the versions of your container images. You could run the versions for DTAP in a single cluster, at different endpoints managed by the cluster software. If necessary, one can still separate a production from a non-production cluster, but this will mostly be from a security perspective, rather than for reasons of stability and availability of the stages in the application's lifecycle.

The entire environment can even be described as code by composition files describing the various elements and containers. For example, the tool Docker Compose uses YML files to describe the various parts in an environment for one or more applications. Transitioning from one environment to the next, say staging to production, can be accomplished by promoting these files.

7. Fit and alignment with distributed architectures
Transitioning to container-based applications fits well with modern architecture styles such as microservices. Large scale and heavily distributed applications benefit from such an architecture, and consequently the use of container technology.

In the aforementioned architectures, individual containers usually contain single processes, such as a service for Web API, website hosting or background processing. This requires one to rethink the design of the application in small, isolated and manageable pieces.

Typically, each of the container images will be running multiple times for failover, robustness and scalability. In addition, there will be many separate container images that together form the complete application. Each of the running container instances is network-separated to build an intrinsically distributed application. The application architecture must take this into account. You should carefully consider the fact that other components use the network for communication. This comes at a price and a risk. The call to another component is not cheap in terms of time, when made over HTTP or TCP compared to an in-process or single-machine call. Also, there is a chance that the process in another container might not be available or reachable. Countermeasures to deal with outages need to be built into the application. Implementation patterns such as Circuit Breaker and Retry become part of the developer's portfolio. It is likely that such an architecture has its effect on the way the teams and company are organized. The benefits of this new architecture type are a very nice alignment with the container approach and with the DevOps teams that build and run the applications. This alignment comes from the isolation of multiple smaller parts in the system, which is designed to deal with potentially short lifetimes or unavailability of the services. The cost of such an architecture and different implementation patterns lies in their complexity and learning curve.

Alex Thissen
Microsoft Visual Studio and Development Technologies MVP

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

MVP Microsoft® Most Valuable Professional

8. Support in application frameworks and tooling

There is an abundance of frameworks to facilitate developing code for components and services to run in containers. The architecture and development model allow a freedom of technology choice for each of the containers, as these are loosely coupled and have no binary or technical dependencies on one another.
There is no specific vendor lock-in when choosing the framework for development. However, a fit-for-purpose selection is highly recommended.
As far as tooling is concerned, the world of containers has converged on Docker as the de facto standard for interacting with container instances, images and hosts. With the standardization around Docker as the container interaction protocol and Docker as a company to deliver the low-level tooling, other companies and software have chosen to align. Development environments offer tooling and integration with the Docker ecosystem. Docker images are used as the image format for container clusters to deploy new components into the cluster. To an increasing extent, web application services are being deployed from Docker images, instead of doing installer or file copy based deployments.

3 Reasons to not use containers

The above-mentioned reasons might be convincing to start with containers tomorrow. As almost always, there is another side to take into account. Containers might not be what is best for you, your applications, or your organization.

1. Container technology is not a silver bullet

Even though containers are popular, especially among developers and operations, it is not the solution for all problems in application delivery and architectures. Switching to containers as a basis for your application will not necessarily solve or prevent performance, stability or scaling issues. Containers do not fit every part of an application or the bigger landscape that it is part of. Careful consideration is required to decide when using containers makes sense and when it doesn't.

2. The benefits of containers come at a price

Creating an application that utilizes containers requires learning the technology, investing time to change the architecture and the deployment strategies. Various design patterns need to be implemented by the developers in order to make a resilient, robust distributed application.
The build and release pipelines have to be changed for the creation and deployment of container images.
This is more complicated than a traditional monolithic application and its lifecycle. The new way of combining development and operations into DevOps involves changing the teams, and aligning the organization with the business domains.
So, while the benefits that an application and the landscape will have from using containers, they do not come free of charge. It means investing before the benefits can be reaped, and there is no guarantee that in the end, the benefits outweigh the costs, particularly for simpler and smaller applications.

3. The application paradigm is moving further

Containers are fairly new and many are still getting their heads around applying them in application development. The world of modern application development is moving on in a rapid pace and other, newer ways of creating distributed applications are emerging. Serverless computing is a nascent form of computing that allows near-infinite scaling and a cost-model that charges by the millisecond consumed. It introduces an even higher level of hardware abstraction, because it does not even require hosts like a cluster would need. Additionally, the way serverless computing is implemented focuses on the actual logic instead of much of the plumbing that container implementations still need. This implies that the costs of building and running applications may be lower when choosing a serverless model.

Final thoughts

Container technology is taking a prominent place in today's approach to application development and hosting. There is a lot to be said and considered for switching to containers as technology for modern cloud-scale applications.

Choose wisely and modernize your IT with containers, ... if it makes sense. </>

What you should know about Windows containers before you start using them

A lot of articles and blogs have already been written about the benefits of using containers for your application delivery[1], also called containerized delivery. While the serverless trend is considered the containers' successor in application delivery, it still is not the best fit for all situations. For example, containerized delivery is a perfect fit for organizations that do not want to move to the cloud (for now) or that have to deal with their existing application landscape. In this article, we'll look at the most important things you as an architect/lead developer have to know about Windows containers before you start using them. If you are interested in more practical manuals about configuring a containerized pipeline, setting up a container registry and other tips and tricks, please have a look at our blog posts[2].

Author Cornell Knulst

Containers as the next step in application delivery

Containerized delivery can be seen as the next step in application delivery for many organizations. If we look at the way in which we have been delivering our software during the past 10 years, we see a shift from manually installing our applications towards an automated way of pushing the applications into production. Driven by the mindset that the reliability, speed and efficiency of delivering our applications to our customers could be improved, various tools and frameworks have been introduced over time, e.g. WiX, PowerShell, PowerShell DSC, Chocolatey, Chef, and Puppet.

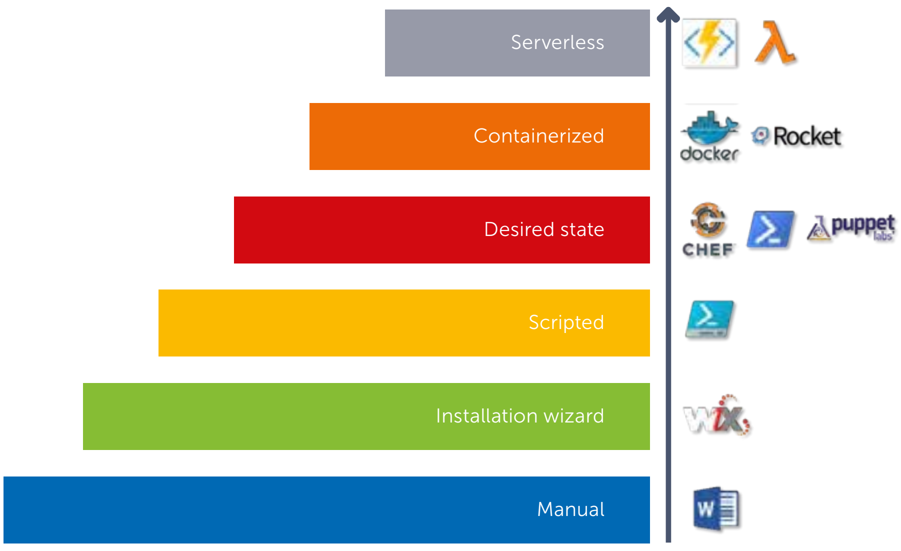


Figure 1 Deployment maturity anno 2017

[1] http://xpir.it/mag4-wincon1, http://xpir.it/mag4-wincon2
[2] https://xpirit.com/blogs

In the earlier years, we baked an MSI by using WiX and the installation of our applications was performed manually by one of our colleagues. Because this way of working was error-prone and it took us a lot of time to find and fix mistakes, we decided to make use of new scripting languages like PowerShell to fully automate the installation of our applications. Although this scripted way of installing our applications brought us more reliability and speed, still some deployments went wrong over different environments (DTAP) because of missing libraries, configurations and tool versions.
So, we decided to make use of Desired State technologies like PowerShell DSC, Chocolatey, Chef, and Puppet to ensure that our environments will end up in the correct state. We succeeded in creating an automated and reliable way of delivering our applications to production, however we still have to reboot machines after de-installations to ensure that all registry keys, caches and file system files are really cleared. Moreover, we experience a lot of waiting- and downtimes caused by (de-)installations and are often not able to run multiple versions of a given application on the same host. That's a pity because we see a lot of VM resources not being consumed by our various applications. Sounds familiar? Deploying your applications in containers can help to solve these issues.

The origin and benefits of containers

Containers are the resulting artifacts of a new level of virtualization that is implemented on Windows. Looking at the history of virtualization, it started with concepts like virtual memory and virtual machines. Containers are the next level of this virtualization trend. Where VMs are a result of hardware virtualization, containers are the result of OS virtualization. Where hardware virtualization lets the VM believe that its hardware resources are dedicated to that instance, OS virtualization lets the container believe that the OS instance is dedicated to that container, although it shares the OS with other containers. The host machine on which different containers can run is called a container host.

Working with containers for your application delivery will give you some benefits as mentioned in the article of Alex Thissen. Instead of installing the application during deployment, containerized delivery will install the application during build and will create a container image out of it. Once you ship your containers to new environments you'll get instant startup times. The image format that is used does not only contain the running application, but also contains a snapshot of the context around the application such as the file system, registry and running processes. This mechanism ensures that our application will run in the same way in different environments. Another benefit is that the container technology helps us to utilize more of the machine resources because it makes it possible to run multiple versions of the same application/conflicting applications on a single host.

How container isolation works

While containers are not VMs, both artifacts support resource governance and an isolated environment[3]. Similar to VMs, each container has an isolated view on its running processes, environment variables, registry and file system. This means that it is possible to change a file within one container while this change is not detected by another container. This isolation characteristic is important to notice. By default, it is not possible to share any files between different containers. Even the container host cannot see the file system changes that are made within the container. The same goes for the registry and environment variables isolation.

The process isolation part is slightly different. While Windows Server Containers cannot see each other's running processes, it is possible to see the running processes of a given Windows Server Containers from the container host. While this is also the case for Linux containers, this is not an appropriate option for multi-tenancy situations where the container host or different containers are not in the same trust boundary. Because of this security implication, Microsoft decided to introduce an extra container type called the Hyper-V container[4]. While it is possible to see the running processes from within the container host for Windows Server Containers, Hyper-V containers run a normal Windows Server Container within a minimalized (utility) Hyper-V "VM". Because of this extra Hyper-V virtualization layer around the normal Windows Server Container it is not possible to see any content of this container type from the container host. Actually, the Hyper-V container is a hybrid model between a VM and a container.

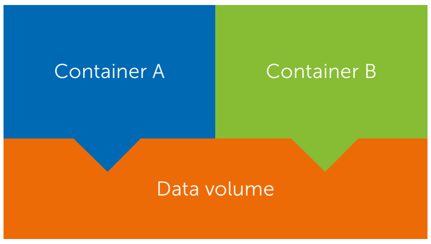
How this impacts the way we share data between our applications

By default, containers are not designed to share any resources. To be able to share data across your containerized applications, you need to store data in remote data storage solutions such as Redis or SQL. However, in some scenarios these solutions are not appropriate because of an existing application architecture you have to deal with. For example, when you have a windows service generating PDFs from files on a given file system location and a website that is putting those files onto this location. How to deal with this inter-container data sharing? We can certainly extend our applications to communicate over HTTP/TCP to exchange important information, but luckily there are also other options available.

Data Volumes One option is that you make use of Data Volumes. Data Volumes are artifacts that differ from containers. They have their own lifecycle and you have to manage them with separate commands. For example, when you delete a container, the Data Volume will still exist. Once you have made a Data Volume, you can map it as a directory on your containers.

[3] http://xpir.it/mag4-wincon3
[4] https://xpir.it/mag4-wincon4

Within the container these mapped directories will bypass the normal Union File System implementation and by doing so, all changes in this directory are always persisted on the file system of the container host. Moreover, changes that are made within this volume are directly available for other containers that consume the same Data Volume. An example script of consuming Data Volumes in your containers can be found below.



```
# Create the DataVolume and give it a name
docker volume create --name loggingvol

# Map the DataVolume with name loggingvol
to directory c:/logging within the container
of Consumer1 and c:/otherlogginglocation
within the container of Consumer2.
docker run -v loggingvol:C:/logging --name
Consumer1 microsoft/windowsservercore
docker run -v loggingvol:C:/otherlogging-
location --name Customer2 microsoft/
windowsservercore
```

Figure 3 Data Volume concept and example script

Data Volume Containers Another option is that you make use of Data Volume containers. Sometimes you need to map multiple Data Volumes for most of the containers you run (for example a logging and data directory). For this purpose, you can make use of Data Volume Containers.
Data Volumes Containers are just containers that you've created and mapped on different Data Volumes. Where normally you must specify the separate volume mappings for each container you run, in the case of Data Volume Containers you just have to add the "--volumes-from" switch to the Docker run command. For each Data Volume that is mapped within the Data Volume Container, Docker will create a new volume mapping on the same file system directories within your consuming containers. While you may expect that deletion of the Volume Container will result in deletion of the Volumes, this is not the case. It just works as normal Data Volumes, except that we have an extra option to easily manage consistency of multiple Volume mappings over different containers.

Volume Plugins By default, Data Volumes are stored on a single container host. But what if I'm running my containers on different container hosts in a cluster and still want to share the data from my Data Volumes on different container hosts? Or what happens with my volume data when my single container host crashes? It's exactly for this reason that Volume Plugins[5] were introduced. Volume Plugins enable you to make Data Volumes hosts independent by integrating their storage with external storage systems such as Amazon EBS. For each Data Volume you can specify whether it should use a different Volume Plugin implementation by specifying the "--driver" switch on volume creation.

Learn about the concepts: images and image layers

When you start building your own containers, you must define what your container will look like. You can achieve this by specifying the various instructions[6] that have to be executed during the build process to define/bake your container. One instruction could be that you copy some files from the build machine to a given location within the container. When using Docker, this file is called a Dockerfile.

During the Docker build process each individual instruction is executed and its resulting file system changes are persisted in an image layer as shown in *Figure 2 - Image layers, Images and Containers*. This image layer will be stored locally on the build server.
The reason for this is that we can have multiple containers that share the same base set of stacked image layers. Instead of rebuilding all these layers again, the Docker build process will look for existing image layers in this local store before it starts building a new layer.

The order in which you specify the instructions makes sense for the image layers that are generated, for example, the Dockerfiles below will create different image layers while they have the same set of instructions. The concept of having a stack of image layers where each image layer represents a set of file system changes is also called a Union File System[7].

```
FROM xpirit/websitecore
MAINTAINER Xpirit <info@xpirit.com>
COPY ./WebDeploy ./WebDeploy
COPY ./CreateIISWebsite.ps1 ./CreateIIS-
Website.ps1

FROM xpirit/websitecore
MAINTAINER Xpirit <info@xpirit.com>
COPY ./CreateIISWebsite.ps1 ./CreateIIS
Website.ps1
COPY ./WebDeploy ./WebDeploy
```

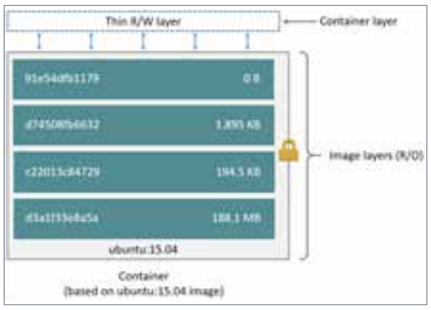


Figure 2 - Image layers, Images and Containers (source: https://docs.docker.com/engine/userguide/storagedriver/imagesandcontainers/)

A container image is delivered at the end of the Docker build process. This image consists of a stacked set of image layers and is the blueprint for all containers you will run on the basis of this image. The difference between a container image and image layers, is that a container image "tags" a given image layer with a user-friendly name. All image layers that are under this tagged layer will be part of the container image. A container image is not a living thing and does not have a state. It is no more than a sealed artifact from which we can create multiple runtime instances: the containers.

[5] https://xpir.it/mag4-wincon5
[6] https://xpir.it/mag4-wincon6
[7] https://xpir.it/mag4-wincon7

To keep track of changes that are made within the container from the moment it was created, each container gets an extra Read/Write layer on top of the stacked image layers of the original container image. Where we might expect a totally different implementation between containers and container images, the only real difference between both objects is just this extra thin R/W layer for each container instance! During the lifecycle of the container, the changes that have been made within the container (actually this R/W layer) are stored temporarily on the container host. Once you delete the container, all those changes are lost. You can prevent this by using the Docker commit command to persist the changes that are part of this thin R/W layer into a separate image layer in the local store. From that moment on you can tag a new image based on this newly created image layer and you can initiate multiple containers based on this image.

Dealing with sensitive information – short peek *Containerized delivery will force you to deal with sensitive information in a different way.*
Once you create a container image out of a running container or from scratch, others can see all of its content just by running a container from that image locally. They can even make use of docker history to see the commands that have run on the container or they can look at the content of the versioned Dockerfile. Hence it is important to be sure that no sensitive information is stored within the container image or Dockerfile. But how to ensure this?
One way of working is that you make use of the Docker Secrets Manager[8] to manage all secrets. Follow our blogs to see how to make this work and what other options are available here.

How these new concepts change the way you'll look at maintaining your applications

Containers are designed as stateless artifacts. You should avoid storing data in your container as much as possible. As mentioned earlier, containers make use of a Read/Write layer to keep track of any changes since the initialization of the container. Once a container is deleted, these changes are lost.
The objective of this implementation is to force users to persist all important changes in container images. By doing this, we can always reproduce a given state and are always able to selectively scale different parts of our system.

Another important characteristic of containers is that they are intended to be immutable. Thanks to the isolation part of containers we don't need to upgrade running applications because we can run different versions of the same container side by side on a single container host. Moreover, because containers also contain the context around the applications, we can be sure that initializing a container on different container hosts will ensure that our application will run in exactly the same way on those different container hosts. Once we accept the stateless character of containers and create container images for all different versions of our applications, there is no longer any need for in-place upgrades. Instead of treating our applications as pets, we can now treat our applications as a flock of applications. If one container gets ill we will not nurse it back to health, but we will remove it and just initiate a new container based on the same genealogical register (the container image).

Containerized delivery will force you to deal with sensitive information in a different way.

[8] https://xpir.it/mag4-wincon8
[9] https://xpir.it/mag4-wincon9
[10] https://xpir.it/mag4-wincon10
[11] https://xpir.it/mag4-wincon11

Reaching Nirvana: Environment-as-Code

Many of us know about Infrastructure-as-Code[9]. The concept of scripting the creation of the infrastructure you need. Containerization will add a layer on top of this Infrastructure-as-Code layer, and this is called Environment-as-Code. Once we have created the necessary container infrastructure like the Container Registry, a cluster of Container Hosts and the VSTS pipeline, it is time to deliver our applications via, and on top of, this infrastructure. Normally we would perform a deployment of our applications individually, and this is still possible within containerized delivery. However, wouldn't it be better from an immutability and quality (repeatability) perspective to always deliver the entire application stack as one? To think about complete environments as deployable items instead of single applications?

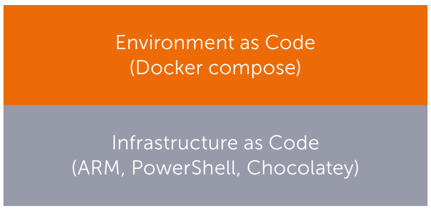


Figure 6 Environment-as-code

This is what we call Environment-as-Code. Not only are your containers immutable and stateless, your complete environment (except your data) should be immutable and stateless as well. And in the same way as we version individual container images, we should also version environment templates. Luckily this is possible with containerized delivery by making use of Docker Compose[10]. Within the compose file you specify the containers (services section), volumes and network related details that are part of your environment. By running the *docker-compose up* command on your docker-compose.yml file, with some environment specific variables specified in an environment[11] file, the Docker engine will eventually create the elements that you have specified in your Docker compose file (including Docker networking).

```
version: '2'
services:
 data:
   image: xpirit/data:1.0.0
   volumes:
     - 'logging:C:/Xpirit/Logging'
     - 'export:C:/Xpirit/Export'

 website:
   image: xpirit/website:1.0.3
   build:
     context: ./Xpirit-website
     dockerfile: Dockerfile
   depends_on:
     - data
   volumes_from:
     - data
   ports:
     - "80:80"
 service:
   image: xpirit/service:2.0.2
   build:
     context: ./Xpirit-agentservice
     dockerfile: Dockerfile
   depends_on:
     - data
   volumes_from:
     - data

volumes:
   logging:
     driver: "local"
   export:
     driver: "local"

networks:
 default:
   external:
     name: nat
```

Figure 7 - Example compose file

Applying Environment-as-Code in practice means that you have to adapt your delivery pipeline based on this concept. All applications/containers will still have a separate commit stage in which compilation, unit testing and acceptance testing on code units takes place. After this commit stage and after running the necessary acceptance tests on our individual containers, you will change the version of the container you just built in the Docker-compose file. Instead of just having a single delivery pipeline for each container, you are also able to deliver the complete environment via a combined delivery pipeline to production.

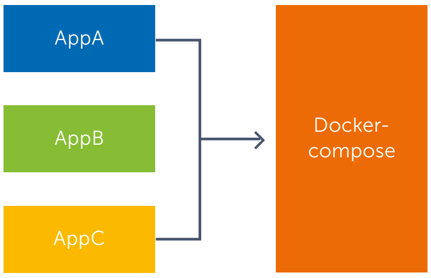


Figure 8 - Combined delivery pipeline

Cornell Knulst

Cornell is a cloud software architect and ALM consultant at Xpirit. He enables companies to deliver software faster, cheaper and better by supplying guidance on adopting DevOps practices and migrating to modern, cloud-native architectures. He is passionate about new technologies and is continuously diving into the latest technologies. He actively maintains a personal blog, speaks regularly at industry events and communities, and regularly publishes articles in national trade magazines.

Where to find more information

We have just looked at a number of important concepts and principles that you need to know before you start performing containerized delivery on the Microsoft stack. There is a lot more to be told about containers and containerized delivery, so please follow our blog posts on https://xpirit.com/blogs or on my personal blog on http://www.solidalm.com to remain up-to-date about working with containers on Windows. A good starting point to learn the necessary commands and tools of working with Docker can be found on https://docs.docker.com/engine/getstarted/. Brighten your life, and start working with containers today! </>

Containerized delivery for .NET workloads on Windows

At the moment one of the big trends in Continuous Delivery is to use containers to speed up value delivery. Containers enable you to run your application in an isolated environment that can be moved between different machines with guaranteed same behavior. Containers can therefore significantly speed up your delivery pipeline, enabling you to deliver features faster to your end users. Now that containers have become part of the Windows Operating System, how can we leverage this to run our existing Windows based .NET workloads like ASP.NET without too many modifications?

Author Marcel de Vries

How can you leverage container innovations, that are already available in the open source and Linux ecosystem? Innovations that enable you to simplify on-demand scaling, fault tolerance and zero downtime deployments of new features. In this article, I will give you a glimpse how you can deploy existing Windows based .NET workloads with Visual Studio Team Services to Azure Container Services, using Kubernetes as the cluster orchestrator.

Why Containers?

Isolation Containers are an isolated, resource controlled, and portable operating environment. They provide a place where an application can run without affecting the rest of the system and without the system affecting the application. If you were inside a container, it looks very much like you are inside a freshly installed physical computer or a virtual machine.

When you create a container, its external dependencies are packed within the container image.
Containers have a layer of protection between host and container and between containerized processes. Containers share the kernel of the host OS. A container relies on the host OS for virtualized access to CPU, memory, network, registry.

Immutability Containers provide the capability of immutability. When you start a container based on a container image, you can make changes to the running environment, but the moment you stop the container and start a new instance of the container, then all changes have been discarded. If you want to capture the state change, then you can after you stopped a container, save the state to a new container image. When you create a new container instance based on the new image only then you will see the changed state.

Less Resource Intensive Running a container compared to running a virtual machine requires very few resources. This is caused by the fact that the operating system is shared. When you start a Virtual Machine, then you boot a whole new operating system on top of the running operating system and you only share the hardware. With containers you share the memory, the disk and the CPU. this means the overhead of starting a container is very low, while it also provides great isolation.

Fast Start-up Time Since running a container requires only few extra resources of the operating system, the startup-time of a container is very fast. The speed of starting a container is comparable to starting a new process.

The only extra things the OS needs to setup is the isolation of the process so it thinks it runs on its own on the machine. This isolation is done at the kernel level and is very fast.

Improve Server Density When you own hardware, then you want to utilize this hardware as good as possible. With virtual machines we made a first step in this direction, by sharing the hardware between multiple virtual machines. Containers take this one step further and enable us to utilize even better the memory, disk and CPU of the hardware available. Since we only consume the memory and CPU we need, we make better use of these resources.
This means less idle running servers hence better utilization of the compute we have. Especially for cloud providers this is very important. The higher server density (the number of things you can do with the hardware you have) the more cost efficient the data-center runs. So it is not strange that Containers are now getting a lot of attention and a lot of new tooling is built around managing and maintaining Containerized solutions.

Why using Container Clusters?

When you want to run your application in production, you want to ensure your customers can keep using your services with as few outages as possible. Therefore, you need to build out an infrastructure that supports concepts like:

- Automatic recovery after an application crash
- Fault tolerance
- Zero downtime deployments
- Resource management cross machines
- Failover

Besides this you want to manage this all in a simple way. This is where container clusters come in to play. The mainstream clusters that are available today are Docker Swarm, DC/OS and Kubernetes. In this article, I will show how to use Kubernetes. Docker Swarm is not really a production grade solution and it seems that Docker is more focused on their Docker data center solution. DC/OS and Kubernetes are the most used clusters in production and Kubernetes already supports windows agents. DC/OS will follow as well soon.

How to Create a cluster in Azure

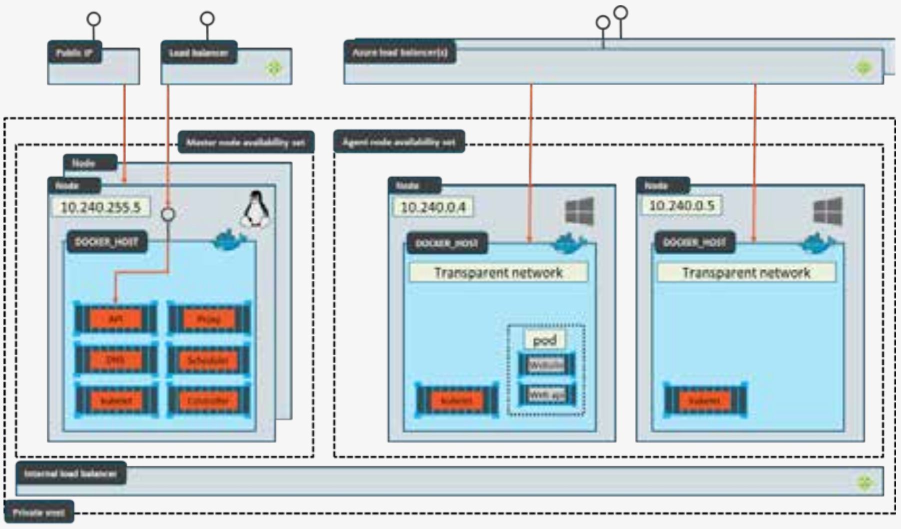
The simplest way to create a container cluster is by using any of the public cloud providers. They all offer clusters that enable you to run your application at scale with a few clicks. Google Cloud Engine provides primarily a cluster based on Kubernetes. Amazon uses as default DC/OS. On Azure you can select which cluster orchestration solution you want to use when you create a cluster. We will have a closer look how we can use Azure.

Portal, Command-line or ACS engine In the portal, you can search for Azure Container Services (ACS) and you will find the option to create a cluster. You have to define the number of Master Nodes and Agents and the Agent Operating system you want to use. Azure supports on Docker Swarm and Kubernetes based clusters, Windows agent nodes. This enables us to deploy our ASP.NET MVC application on Windows containers in a cluster.
From a Continuous delivery perspective, we always prefer to enable the creation of the infrastructure as part of our delivery pipeline So creating it via the command line is the preferred way of doing this, since it provides us the ability to repeat the steps we have taken and check it in as a provisioning script for future use to set up a new environment.

Azure has a new command-line interface 2.0 that supports the creation of ACS clusters. The following command can be used to create a cluster:

```
az acs create --orchestrator-type=kubernetes --resource-group
myresourcegroup --name= my-acs-cluster-name --dns-prefix=
some-unique-value
```

The moment we create a cluster we will have a setup that looks as follows:



Master nodes When we look at the cluster that will be created for us, you will find that the Master Nodes are Linux based virtual machines. The masters are responsible for managing the cluster and scheduling the containers based on the resource constraints we give to the deployment definitions. When you define a deployment you send commands to the master, which in its turn will schedule the containers to be run on the on the agents. The way we communicate with the master is through a command line tool called *kubectl*. This command line tool issues the commands against the API server running on the master nodes.
The master nodes run a set of containers that support the cluster like e.g. the cluster DNS service and the scheduler engine.

Agent nodes The agents run the kubelet containers, that manages the agent communication and interactions with the master. The way the master communicates to the agents is via the local network that is not exposed outside the cluster. If we want to expose a service (one or more containers) to the outside world, we can do this with a simple *kubectl command, kubectl expose deployment <name of deployment> --port=port#* which in its turn will expose the deployment via the Azure load balancer to the outside world. The cluster will manage the configuration and creation of the required load-balancer(s), the allocation of public IP addresses and configuring the load-balancing rules.

Deployments , Pods and Services
In a deployment, you describe a combination of Docker images you want to run in your cluster. This combination of images, including the shared storage options and run options, defines a Pod. A pod is the implicitly defined logical unit of container instance management. When container instances are created for the images in a pod, they will always run together on the same node. Let's say you have an application that consists of two parts: a web API and a local cache, that are only effective when running on the same node. To accomplish this, you can define a deployment template that includes the two images from the command-line or in a yaml file.
The deployment also specifies how many instances of pods will be started. By default, this is a single instance of a pod. For fault tolerance and scaling it is possible to increase the so-called replica count of your deployment to start multiple copies of the pods.
The moment you start the deployment, the cluster is responsible for deploying the pods to the various nodes and balancing the resources in the cluster.

Every container instance created in a pod will request an IP address from the DHCP Server in the local cluster network. Additionally, each pod will get an internal DNS record based on the deployment name. The master node acts as both DHCP and DNS Server.

Container instances become reachable from anywhere in the cluster, based on their DNS name and exposed ports. When you expose a deployment, the cluster will connect the container instances in pods to the external load-balancer, so they are reachable from outside the cluster as well.
Kubernetes uses the notion of a Service as the abstraction that defines the logical set of Pods and the policies to expose the endpoints we need. This is sometimes referred to as the micro-service. You can list which endpoints in your cluster are exposed to the outside world by running the command:

```
Kubectl get services
```

This then shows the list of services and the endpoint details like ip address and port on which the workloads are reachable.

Zero downtime replacement of deployments

When running applications at internet scale, you want to be able to deploy new features to the end users without any downtime of the application.
Using a Kubernetes cluster makes this possible with the concept of a rolling update. Rolling updates enable you to update your container images in your container registry and then give a command to update the container images of the running container instances. This can be done with a single command-line:

```
kubectl set image deployment
<nameofdeployment> <nameofimage>=
<reponame/newimagename>
```

In this command, you can precisely specify which image you want to change, since a deployment can contain multiple images.
The steps taken are the following:

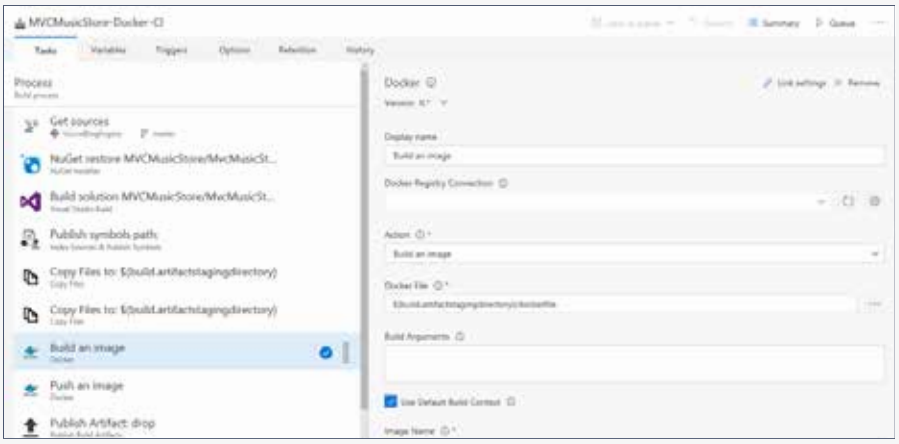
- Spin up new pods on the various nodes.
- Drain traffic to the old pods
- Gate traffic to the new pods

In these steps the cluster ensures we always keep the minimum set of pods up and running so we can guarantee we can keep handling traffic while the deployment is in transit. The minimum set of replicas that always need to be up, can be specified in the deployment when created.

Deployments using Visual Studio Team Services (VSTS)

To ensure a robust, repeatable and reliable way of deploying your application to the cluster, you can use the build and release capabilities of VSTS. When you deploy a new feature to your application in the cluster, you will go through two primary phases. Phase 1, build, test and publish the container image. Phase 2, run the new image in the various test environments and finally deploy it to the cluster, using the zero-downtime deployment capability.

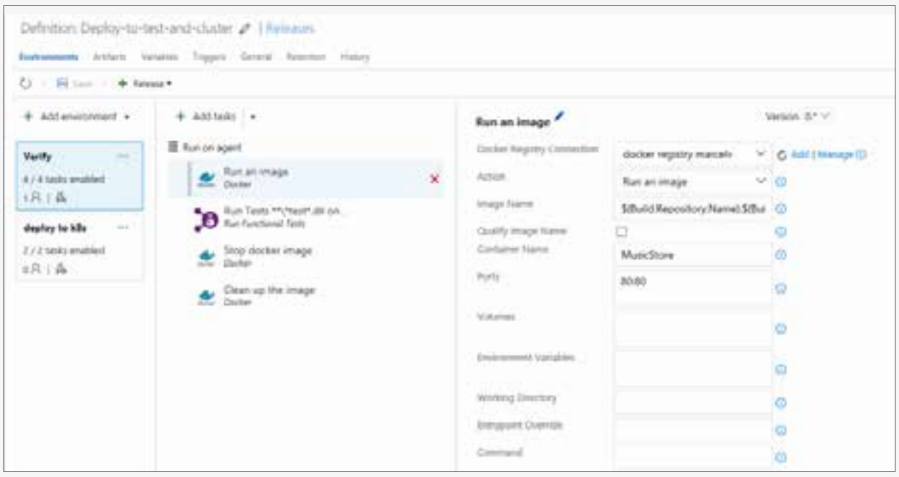
Phase 1, build the container image
In phase 1 we use the VSTS build infrastructure. For this we simply build our container based on a docker file we check into source control. In the build you define a step where we build the image using the docker file that picks up the build artifacts from your application. (dll's, configuration, web content, etc) You can see an example here in the screenshot:



Get sources
NuGet restore MVCMusicStore/MvcMusicSt...
Build solution MVCMusicStore/MvcMusicSt...
Publish symbols path
Copy Files to: $(build.artifactstagingdirectory)
Copy Files to: $(build.artifactstagingdirectory)
Build an image
Push an image
Publish Artifact: drop

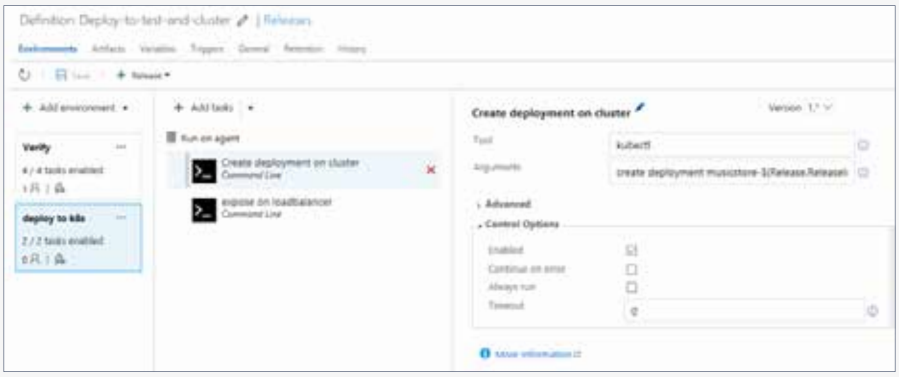
Display name: Build an image
Docker Registry Connection
Action: Build an image
Docker File
Build Arguments
Use Default Build Context
Image Name

Phase 2, test and deploy to production In this phase we use the Release management part of VSTS, that uses the same agent infrastructure as the build. You can define a set of environments where you first validate the new feature(s). The moment you have gathered enough evidence and confidence the new application runs as expected you can then move to the deployment environment and deploy to production. In VSTS you can specify tasks that need to be run in each environment. Below you can see the series of steps you can use to test the image you just created in the first phase, by running Docker tasks that start the container. Next, you see a task for testing the running container and the final step is to stop the container.


Definition: Deploy-to-test-and-cluster | Releases
Environments Artifacts Variables Triggers General Retention History
Save Release
Add environment
Verify
4 / 4 tasks enabled
deploy to k8s
2 / 2 tasks enabled
Add tasks
Run on agent
Run an image Docker
Run Tests ***test*.dll on... Run Functional Tests
Stop docker image Docker
Clean up the image Docker
Run an image
Version: 0.*
Add | Manage

Docker Registry Connection: docker registry marcelv
Action: Run an image
Image Name: $(Build.Repository.Name):$(Bui
Qualify Image Name: ☐
Container Name: MusicStore
Ports: 80:80
Volumes:
Environment Variables:
Working Directory:
Entrypoint Override:
Command:

Deploying to production is done in the production environment with a set of tasks. These tasks execute the previous described command-line tool kubectl. For this to work you do need to install the kubectl binaries on the agent machine and add location to the %path% system environment variable. From that moment on, you can issue any kubectl command to the cluster to create or update deployments. The flow for the deploy to production is shown in the next screenshot:


Definition: Deploy-to-test-and-cluster | Releases
Environments Artifacts Variables Triggers General Retention History
Save Release
Add environment
Verify
4 / 4 tasks enabled
deploy to k8s
2 / 2 tasks enabled
Add tasks
Run on agent
Create deployment on cluster Command Line
expose on loadbalancer Command Line
Create deployment on cluster
Version: 1.*
More information

Tool: kubectl
Arguments: create deployment musicstore-$(Release.ReleaseI
Advanced
Control Options
Enabled: ☑
Continue on error: ☐
Always run: ☐
Timeout: 0

Improve speed of value delivery

In this article I introduced you to the concept of containerized delivery using containers and Azure ACS. It is now possible to run your existing ASP.NET applications in containers because Microsoft added containers and Docker support to their operating system Windows Server 2016 and Windows 10. The open source cluster orchestrators like Kubernetes, DC/OS and Docker Swarm also enabled support for Windows containers, unlocking containerized delivery now also in the Windows ecosystem. Because containers provide a mechanism to very easily move them around in different environments, guaranteeing exact same behavior, it now becomes much simpler and faster to deploy to both your test and production environment. When you add to this the flexibility of scaling, fault tolerance and zero downtime deployments with clusters, then you can really improve the speed of feature delivery to your customers.

Marcel de Vries
Microsoft Regional Director, Microsoft Visual Studio and Development Technologies MVP, Xamarin MVP

Marcel is the co-founder and CTO of Xpirit. Marcel spends most of his time learning and teaching new emerging technologies, a shift in mindset, and a new way of working to help organizations produce value faster. Helping people and organizations transform to become more innovative and productive is his passion. You can find him speaking regularly at industry events and communities around the world, e.g. Visual Studio Live, Tech Days, Dev Intersection, and Techorama.

MVP Microsoft® Most Valuable Professional | Microsoft Regional Director

Conclusion

Containerized delivery is now also possible for your existing workloads on Windows like ASP.NET. You can utilize Azure ACS clusters with a cluster orchestrator like Kubernetes to manage your workloads at scale with much more ease of deployment than in the past. With containerized delivery we simplify the build, test and deployment pipelines and we significantly improve our delivery cycle time. </>

Cloud transformation done right!

We are Xpirit

Experts in new Microsoft Technology

www.xpirit.com/cloud

PROUDLY PART OF XEBIA GROUP

Xpirit

Think ahead. Act now.

www.xpirit.com

Think ahead.
Act now.

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