

Please Release Me

Search here ...

Continuous Delivery,
DevOps, ALM and IoT in a
Mostly Microsoft Azure
World

Deploy a Dockerized ASP.NET Core Application to Azure Kubernetes Service Using a VSTS CI/CD Pipeline: Part 4

Posted by Graham Smith on September 11, 2018 | 0 Comments

In this blog post series I'm working my way through the process of deploying and running an ASP.NET Core application on Microsoft's hosted Kubernetes environment. These are the links to the full series of posts to date:

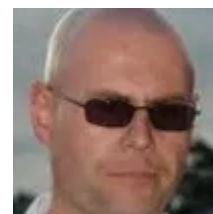
- [Deploy a Dockerized ASP.NET Core Application to Kubernetes on Azure Using a VSTS CI/CD Pipeline: Part 1](#)
- [Deploy a Dockerized ASP.NET Core Application to Kubernetes on Azure Using a VSTS CI/CD Pipeline: Part 2](#)
- [Deploy a Dockerized ASP.NET Core Application to Azure Kubernetes Service Using a VSTS CI/CD Pipeline: Part 3](#)
- Deploy a Dockerized ASP.NET Core Application to Azure Kubernetes Service Using a VSTS CI/CD Pipeline: Part 4 (this post)

In this post I take a look at application monitoring and health. There are several options for this however since I'm pretty much all-in with the Microsoft stack in this blog series I'll stick with the Microsoft offering which is [Azure Application Insights](#). This posts builds on previous posts,

About the Author

Dr Graham Smith is a former research scientist who got bitten by the programming and database bug so badly that in 2000 he changed careers to become a full-time software developer.

After spending 12 years as a .NET



/ SQL Server software engineer Graham spent three years leading a major CI/CD pipeline implementation using Microsoft technologies followed by three years in senior IT leadership roles leading wider DevOps initiatives. Graham currently works for **DevOpsGroup** as a Senior DevOps

particularly [Part 3](#), so please do consider working through at least Part 3 before this one.

In this post, I continue to use my MegaStore sample application which has been upgraded to .NET Core 2.1, in particular with reference to the `csproj` file. This is important because it affects the way Application Insights is configured in the ASP.NET Core web component. See [here](#) and [here](#) for more details. All my code is in my GitHub repo and you can find the starting code [here](#) and the finished code [here](#).

Understanding the Application Insights Big Picture

Whilst it's very easy to get started with Application Insights, configuring it for an application with multiple components which gets deployed to a continuous delivery pipeline consisting of multiple environments running under Kubernetes requires a little planning and a little effort to get everything configured in a satisfactory way. As of the time of writing this isn't helped by the presence of Application Insights documentation on both docs.microsoft.com and github.com ([ASP.NET Core](#) | [Kubernetes](#)) which sometimes feels like it's conflicting, although it's nothing that good old fashioned detective work can't sort out.

The high-level requirements to get everything working are as follows:

1. A mechanism is needed to separate out telemetry data from the different environments of the continuous delivery pipeline. Application Insights sends telemetry to a 'bucket' termed an **Application Insights Resource** which is identified by a unique instrumentation key. Separation of telemetry data is therefore achieved by creating an individual Application Insights Resource, each for the development environment and the different environments of the delivery pipeline.
2. Each component of the application that will send telemetry to an Application Insights Resource needs configuring so that it can be supplied with the instrumentation key for the Application Insights Resource for the environment the application is running in. This is a coding issue and there are lots of ways to solve it, however in the MegaStore sample application this is achieved through a helper class in the `MegaStore.Helper` library that receives the instrumentation key as an environment variable.
3. The `MegaStore.Web` and `MegaStore.SaveSaleHandler` components need configuring for both the core and Kubernetes elements of Application Insights and a mechanism to send the telemetry back to Azure with the actual name of the component rather than a name that Application Insights has chosen.
4. Each environment needs configuring to create an instrumentation key environment variable for the Application Insights Resource that has been created for that environment. In development this is achieved through hard-

Transformation

Consultant and in his spare time is a **CodeClub** volunteer helping to teach kids programming on the Raspberry Pi platform. The opinions expressed here are Graham's and not necessarily those of the DevOpsGroup.

Simple-Talk Awards 2015/16



Blog Series

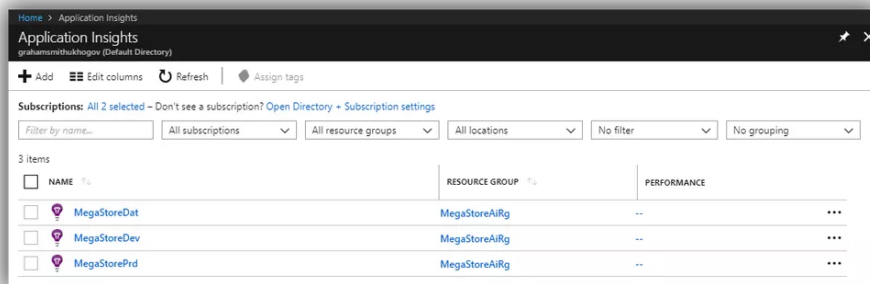
Internet of Things
Continuous Delivery with Containers
Continuous Delivery with TFS / VSTS
Continuous Delivery with TFS
Continuous Delivery with VSO
ALM Practices
Getting Started
Tools, Tips and Tricks

coding the instrumentation key in `docker-compose.override.yaml`. In the deployment pipeline it's achieved through a VSTS task that creates a Kubernetes config map that is picked up by the Kubernetes deployment configuration.

That's the big picture—let's get stuck in to the details.

Creating Application Insights Resources for Different Environments

In the Azure portal follow [these](#) slightly outdated instructions (Application Insights is currently found in Developer Tools) to create three Application Insights Resources for the three environments: DEV, DAT and PRD. I chose to put them in one resource group and ended up with this:



For reference there is a dedicated [Separating telemetry from Development, Test, and Production](#) page in the official Application Insights documentation set.

Configure MegaStore to Receive an Instrumentation Key from an Environment Variable

As explained above this is a specific implementation detail of the MegaStore sample application, which contains an `Env` class in `MegaStore.Helper` to capture environment variables. The amended class is as follows:

```
using System;
using System.Collections.Generic;

namespace MegaStore.Helper
{
    // This code is modified from https://github.com/sixeyed/docker-on-windows
    public class Env
    {
        private static Dictionary<string, string> _Values = new Dictionary<
string, string>();

        public static string MessageQueueUrl { get { return Get(
"MESSAGE_QUEUE_URL"); } }
```

Follow via Email

Follow via RSS

RSS - Posts

RSS - Comments

Recent Posts

Create an Azure DevOps
Services Self-Hosted
Agent in Azure Using
Terraform, Cloud-init—
and Azure DevOps
Pipelines!
Deploy a Dockerized
ASP.NET Core
Application to Azure
Kubernetes Service
Using a VSTS CI/CD
Pipeline: Part 4
Upgrade a Dockerized
ASP.NET Core

```

        public static string DbConnectionString { get { return Get(
"DB_CONNECTION_STRING"); } }

        public static string AppInsightsInstrumentationKey { get { return
Get("APP_INSIGHTS_INSTRUMENTATION_KEY"); } }

        private static string Get(string variable)
        {
            if (!_Values.ContainsKey(variable))
            {
                var value = Environment.GetEnvironmentVariable(variable);
                _Values[variable] = value;
            }
            return _Values[variable];
        }
    }
}

```

Obviously this class relies on an external mechanism creating an environment variable named APP_INSIGHTS_INSTRUMENTATION_KEY. Consumers of this class can reference MegaStore.Helper and call Env.AppInsightsInstrumentationKey to return the key.

Configure MegaStore.Web for Application Insights

If you've upgraded an ASP.NET Core web application to 2.1 or later as detailed earlier then the core of Application Insights is already 'installed' via the inclusion of the Microsoft.AspNetCore.All meta package so there is nothing to do. You will need to add Microsoft.ApplicationInsights.Kubernetes via NuGet—at the time of writing it was in beta (1.0.0-beta9) so you'll need to make sure you have told NuGet to include prereleases.

In order to enable Application Insights amend BuildWebHost in Program.cs as follows:

```

public static IWebHost BuildWebHost(string[] args) =>
    WebHost.CreateDefaultBuilder(args)
        .UseStartup<Startup>()
        .UseApplicationInsights(Env.AppInsightsInstrumentationKey)
        .Build();

```

Note the way that the instrumentation key is passed in via Env.AppInsightsInstrumentationKey from MegaStore.Helper as mentioned above.

Telemetry relating to Kubernetes is enabled in ConfigureServices in Startup.cs as follows:

```

public void ConfigureServices(IServiceCollection services)
{
    services.EnableKubernetes();
    services.AddMvc();
}

```

Application to the Latest

Version of .NET Core

Deploy a Dockerized

ASP.NET Core

Application to Azure

Kubernetes Service

Using a VSTS CI/CD

Pipeline: Part 3

Deploy a Dockerized

ASP.NET Core

Application to Kubernetes

on Azure Using a VSTS

CI/CD Pipeline: Part 2

Tags

Agile ALM **Application Insights** automated testing
Azure Azure Automation
Azure CLI **Azure Resource Manager**
Containers
Continuous Delivery
Continuous Integration
DevOps **Docker** **Git**
GitHub IIS JSON kubectl
Kubernetes **Linux**
Microsoft Test Manager MSDN
PowerShell
PowerShell DSC
Python **Raspberry Pi** Release

```
services.AddSingleton<ITelemetryInitializer,
CloudRoleTelemetryInitializer>();
}
```

Note also that a `CloudRoleTelemetryInitializer` class is being initialised. This facilitates the setting of a custom `RoleName` for the component, and requires a class to be added as follows:

```
using Microsoft.ApplicationInsights.Channel;
using Microsoft.ApplicationInsights.Extensibility;

namespace MegaStore.Web
{
    public class CloudRoleTelemetryInitializer : ITelemetryInitializer
    {
        public void Initialize(ITelemetry telemetry)
        {
            telemetry.Context.Cloud.RoleName = "MegaStore.Web";
        }
    }
}
```

Note here that we are setting the `RoleName` to `MegaStore.Web`. Finally, we need to ensure that all web pages return telemetry. This is achieved by adding the following code to the end of `_ViewImports.cshtml`:

```
@inject Microsoft.ApplicationInsights.AspNetCore.JavaScriptSnippet
JavaScriptSnippet
```

and then by adding the following code to the end of the `<head>` element in `_Layout.cshtml`:

```
@Html.Raw(JavaScriptSnippet.FullScript)
```

Configure MegaStore.SaveSaleHandler for Application Insights

I'll start this section with a warning because at the time of writing the latest versions of `Microsoft.ApplicationInsights` and `Microsoft.ApplicationInsights.Kubernetes` didn't play nicely together and resulted in dependency errors. Additionally the latest version of `Microsoft.ApplicationInsights.Kubernetes` was missing the `KubernetesModule.EnableKubernetes` class described in the [documentation](#) for making Kubernetes work with Application Insights. The Kubernetes bits are still in beta though so it's only fair to expect turbulence. The good news is that with a bit of experimentation I got everything working by installing NuGet packages `Microsoft.ApplicationInsights` (2.4.0) and `Microsoft.ApplicationInsights.Kubernetes` (1.0.0-beta3). If you try this long after publication date things will have moved on but this combination works with this initialisation code in `Program.cs`:

```
/*
Requires these using statements:
using Microsoft.ApplicationInsights;
using Microsoft.ApplicationInsights.Extensibility;
```

Management
Release
Management
for Visual
Studio Selenium

SonarQube SQL Server

SQL Server Reporting Services

SSDT Team
Foundation
Server

Test Impact

Analysis TFBUILD Version

Control Virtual Machine

Visual Studio
Visual Studio
Online Visual
Studio Team
Services vsts

Windows Server Windows

Subsystem for Linux

WordPress

```
using Microsoft.ApplicationInsights.Kubernetes;
*/

class Program
{
    private static TelemetryConfiguration configuration = new
    TelemetryConfiguration(Env.AppInsightsInstrumentationKey);

    static void Main(string[] args)
    {
        configuration.TelemetryInitializers.Add(new
    CloudRoleTelemetryInitializer());
        KubernetesModule.EnableKubernetes(configuration);

        TelemetryClient client = new TelemetryClient(configuration);
        client.TrackTrace("Some message");
    }
}
```

Please do note that this a *completely* stripped down Program class to just show how Application Insights and the Kubernetes extension is configured. Note again that this component uses the CloudRoleTelemetryInitializer class shown above, this time with the RoleName set to MegaStore.SaveSaleHandler. What I don't show here in any detail is that you can add lots of client.Track* calls to generate rich telemetry to help you understand what your application is doing. The [code](#) on my GitHub repo has details.

Configure the Development Environment to Create an Instrumentation Key Environment Variable

This is a simple matter of editing docker-compose.override.yaml with the new APP_INSIGHTS_INSTRUMENTATION_KEY environment variable and the instrumentation key for the corresponding Application Insights Resource:

```
version: '3.5'

services:

    message-queue:
        image: nats:linux
        networks:
            - ms-net

    megastore.web:
        environment:
            - ASPNETCORE_ENVIRONMENT=Development
            - MESSAGE_QUEUE_URL=nats://message-queue:4222
            - APP_INSIGHTS_INSTRUMENTATION_KEY=fba89ed9-z023-48f0-a7bb-6279ba6b5c8
7
        ports:
            - 80
        depends_on:
            - message-queue
        networks:
```



```

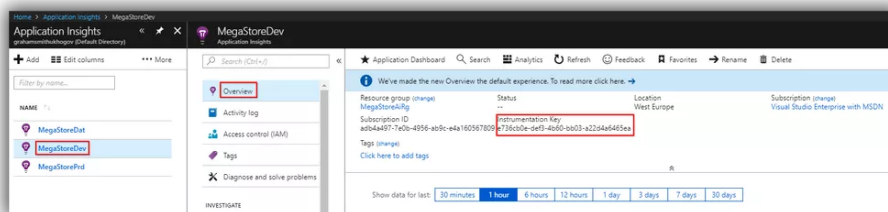
- ms-net

megastore.savesalehandler:
  environment:
    - MESSAGE_QUEUE_URL=nats://message-queue:4222
    - APP_INSIGHTS_INSTRUMENTATION_KEY=fba89ed9-z023-48f0-a7bb-6279ba6b5c8
7
  env_file:
    - db-credentials.env
  depends_on:
    - message-queue
  networks:
    - ms-net

networks:
  ms-net:

```

Make sure you don't just copy the code above as the *actual* key needs to come from the Application Insights Resource you created for the DEV environment, which you can find as follows:



Configure the VSTS Deployment Pipeline to Create Instrumentation Key Environment Variables

The first step is to amend the two Kubernetes deployment files (`megastore-web-deployment.yaml` and `megastore-savesalehandler-deployment.yaml`) with details of the new environment variable in the respective `env` sections:

```

- name: APP_INSIGHTS_INSTRUMENTATION_KEY
  valueFrom:
    configMapKeyRef:
      name: appinsights.env
      key: APP_INSIGHTS_INSTRUMENTATION_KEY

```

Now in VSTS:

1. Create variables called `DatAppInsightsInstrumentationKey` and `PrdAppInsightsInstrumentationKey` scoped to their respective environments and populate the variables with the respective instrumentation keys.
2. In the task lists for the DAT and PRD environments clone the `Delete ASPNETCORE_ENVIRONMENT` config map and `Create ASPNETCORE_ENVIRONMENT` config map tasks and amend them to

work with the new APP_INSIGHTS_INSTRUMENTATION_KEY environment variable configured in the *.deployment.yaml files.

Generate Traffic to the MegaStore Web Frontend

Now the real fun can begin! Commit all the code changes to trigger a build and deploy. The clumsy way I'm having to delete an environment variable and then recreate it (to cater for a changed variable name) will mean that the release will be amber in each environment for a couple of releases but will hopefully eventually go green. In order to generate some interesting telemetry we need to put one of the environments under load as follows:

1. Find the public IP address of MegaStore.Web in the PRD environment by running `kubectl get services --namespace=prd`:

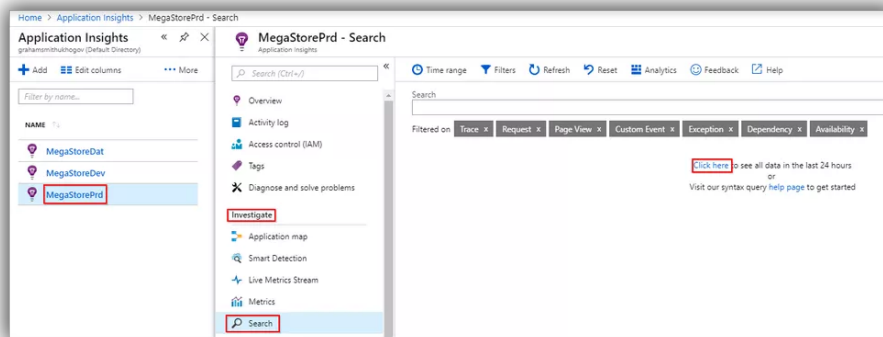
```
C:\Users\Graham>kubectl get services --namespace=prd
NAME                                TYPE        CLUSTER-IP    EXTERNAL-IP    PORT(S)          AGE
megastore-web-service              LoadBalancer  10.0.236.22    23.97.208.183  80:31518/TCP     96d
message-queue-service              ClusterIP     10.0.208.128   <none>         4222/TCP,6222/TCP,8222/TCP 96d
```

2. Create a PowerShell (ps1) file with the following code (using your own IP address of course):

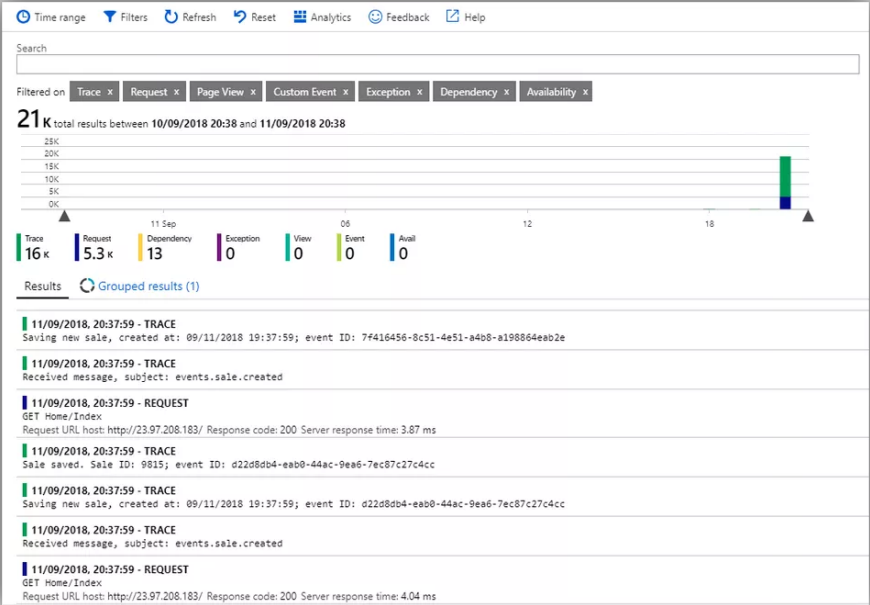
```
while ($true)
{
    (New-Object Net.WebClient).DownloadString("http://23.97.208.183/")
    Start-Sleep -Milliseconds 5
}
```

3. Run the script (in Windows PowerShell ISE for example) and as long as the output is white you know that traffic is getting to the website.

Now head over to the Azure portal and navigate to the Application Insights Resource that was created for the PRD environment earlier and under **Investigate** click on **Search** and then **Click here** (to see all data in the last 24 hours):



You should see something like this:



Hurrah! We have telemetry! However the icing on the cake comes when you click on an individual entry (a trace for example) and see the Kubernetes details that are being returned with the basic trace properties:

TRACE		
Trace Properties		Show less
Event time	11/09/2018, 20:37:59	...
Device type	PC	...
Message	Saving new sale, created at: 09/11/2018 19:37:59; event ID: 7f416456-8c51-4e51-a4b8-a198864eab2e	...
SDK version	dotnet:2.4.0-32153	...
Client IP address	0.0.0.0	...
Sample rate	1	...
City	Amsterdam	...
State or province	North Holland	...
Country or region	Netherlands	...
Cloud role name	MegaStore.SaveSaleHandler	...
Telemetry type	trace	...
Custom Data		
Kubernetes.ReplicaSet.Name	megastore-savesalehandler-deployment-6dd5789659	...
Kubernetes.Deployment.Name	megastore-savesalehandler-deployment	...
Kubernetes.Container.Name	megastoresavesalehandler	...
Kubernetes.Container.ID	ceb93bfc27c4f1f4a8672872c98f52a18b1ea0f7c7b2e0cb906a13e995fd934	...
Kubernetes.Pod.Labels	app:megastore-savesalehandler,pod-template-hash:2881345215	...
Kubernetes.Node.Name	aks-agentpool-34239724-0	...
Kubernetes.Pod.Name	megastore-savesalehandler-deployment-6dd5789659-7xmpk	...
Kubernetes.Node.ID	aef3911c-6a76-11e8-9958-6eb4cb21a	...
Kubernetes.Pod.ID	84ee2cbb-b5f3-11e8-8a3c-0a4c2c81b710	...

Until Next Time

It's taken my quite a lot of research and experimentation to get to this point so that's it for now! In this post I showed you how to get started monitoring

your Dockerized .NET Core 2.1 applications running in AKS using Application Insights. The emphasis has been very much on getting started though as Application Insights is a big beast and I've only scratched the surface in this post. Do bear in mind that some of the NuGets used in this post are in beta and some pain is to be expected.

As I publish this blog post [VSTS has had a name change to Azure DevOps](#) so that's the title of this series having to change again!

Cheers—Graham

Share this:

Share 0

Share

Tweet

G+






6

Save

Email

Print

Posted in **Continuous Delivery with Containers**

Tags:  Application Insights  Azure DevOps  Kubernetes
 Visual Studio Team Services  VSTS

0 Comments

pleasereleaseme

 Login ▾ Recommend Tweet Share

Sort by Best ▾

LOG IN WITH

OR SIGN UP WITH DISQUS 

Be the first to comment.

ALSO ON PLEASERELEASEME

Continuous Delivery with TFS / VSTS – Instrument for

4 comments • 2 years ago

Graham Smith — Glad all's well Harley and thanks for posting back!

Continuous Delivery with Containers – Use Visual

8 comments • 2 years ago

Graham Smith — Hi GiorgiGlad you liked the article! I see you are using TFS 2017, however I

Continuous Delivery with Containers – Azure CLI

6 comments • 2 years ago

Graham Smith — Thanks for the update!

Ubiquiti WiFi: How I Got Started with this Fantastic Kit

6 comments • a year ago

Chris Buechler — USG does DNS by default, dnsmasq is included and client hostnames

 **Subscribe**  **Add Disqus to your site** Add DisqusAdd

Powered by WordPress, Theme by Andrew Dyer