Please Release Me

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

Search here ...

Deploy a Dockerized ASP.NET Core Application to Azure Kubernetes Service Using a VSTS CI/CD Pipeline: Part 3

Posted by **Graham Smith** on **May 24, 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. Formerly known as Azure Container Service (AKS), it has recently been renamed Azure Kubernetes Service, which is why the title of my blog series has changed slightly. In previous posts in this series I covered the key configuration elements both on a developer workstation and in Azure and VSTS and then how to actually deploy a simple ASP.NET Core application to AKS using VSTS. This is 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 (this post)

In this post I introduce MegaStore (just a fictional name), a more complicated ASP.NET Core application (in the sense that it has more

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

moving parts), and I show how to deploy MegaStore to an AKS cluster using VSTS. Future posts will use MegaStore as I work through more advanced Kubernetes concepts. To follow along with this post you will need to have completed the following, variously from parts 1 and 2:

- Development Workstation Configuration
- Create Services in Microsoft Azure
- Create VSTS Endpoints
- Environments and Namespaces

Introducing MegaStore

MegaStore was inspired by Elton Stoneman's evolution of NerdDinner for his excellent book Docker on Windows, which I have read and can thoroughly recommend. The concept is a sales application that rather than saving a 'sale' directly to a database, instead adds it to a message queue. A handler monitors the queue and pulls new messages for saving to an Azure SQL Database. The main components are as follows:

- MegaStore.Web—an ASP.NET Core MVC application with a **CreateSale** method in the HomeController that gets called every time there is a hit on the home page.
- NATS message queue—to which a new sale is published.
- MegaStore.SaveSalehandler—a .NET Core console application that monitors the NATS message queue and saves new messages.
- Azure SQL Database—I recently heard Brendan Burns comment in a podcast that hardly anybody designing a new cloud application should be managing storage themselves. I agree and for simplicity I have chosen to use Azure SQL Database for all my environments including development.

You can clone MegaStore from my GitHub repository here.

In order to run the complete application you will first need to create an Azure SQL Database. The easiest way is probably to create a new database (also creates a server at the same time) via the portal and manage with SQL Server Management Studio. The high-level procedure is as follows:

1. In the portal create a new database called **MegaStoreDev** and at the same time create a new server (name needs to be unique). To keep costs low I start with the **Basic** configuration knowing I can scale up and down as required.
2. Still in the portal add a client IP to the firewall so you can connect from your development machine.
3. Connect to the server/database in SSMS and create a new table called dbo.Sale:

```
SET ANSI_NULLS ON
GO
```

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

```
SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Sale](
    [SaleID] [bigint] IDENTITY(1001,1) NOT NULL,
    [CreatedOn] [datetime] NOT NULL,
    [Description] [varchar](100) NOT NULL
) ON [PRIMARY]
GO
```

4. In **Security** > **Logins** create a **New Login** called **sales_user_dev**, noting the password.
5. In **Databases** > **MegaStoreDev** > **Security** > **Users** create a **New User** called **sales_user** mapped to the **sales_user_dev** login and with the **db_owner role**.

In order to avoid exposing secrets via GitHub the credentials to access the database are stored in a file called **db-credentials.env** which I've not committed to the repo. You'll need to create this file in the **docker-compose** project in your VS solution and add the following, modified for your server name and database credentials:

```
DB_CONNECTION_STRING=Server=tcp:megastore.database.windows.net,1433;Initial
Catalog=MegaStoreDev;Persist Security Info=False;User ID=sales_user_dev;Pass
word=mystrongpwd;MultipleActiveResultSets=False;Encrypt=True;TrustServerCert
ificate=False;Connection Timeout=30;
```

If you are using version control make sure you exclude **db-credentials.env** from being committed.

With **docker-compose** set as the startup project and Docker for Windows running set to Linux containers you should now be able to run the application. If everything is working you should be able to see sales being created in the database.

To understand how the components are configured you need to look at **docker-compose.yml** and **docker-compose-override.yml**. Image building is handled by **docker-compose.yml**, which can't have anything else in it otherwise VSTS complains if you want to use the compose file to build the images. The configuration of the components is specified in **docker-compose-override.yml** which gets merged with **docker-compose.yml** at run time. Notice the **k8s** folder. This contains the configuration files needed to deploy the application to AKS.

By now you may be wondering if MegaStore should be running locally under Kubernetes rather than in Docker via docker-compose. It's a good question and the answer is probably yes. However at the time of writing there isn't a great story to tell about how Visual Studio integrates with Kubernetes on a developer workstation (ie to allow debugging as is possible with Docker) so I'm purposely ignoring this for the time being. This will change over time though, and I will cover this when I think there is more to tell.

Follow via Email

Email Address

Subscribe

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

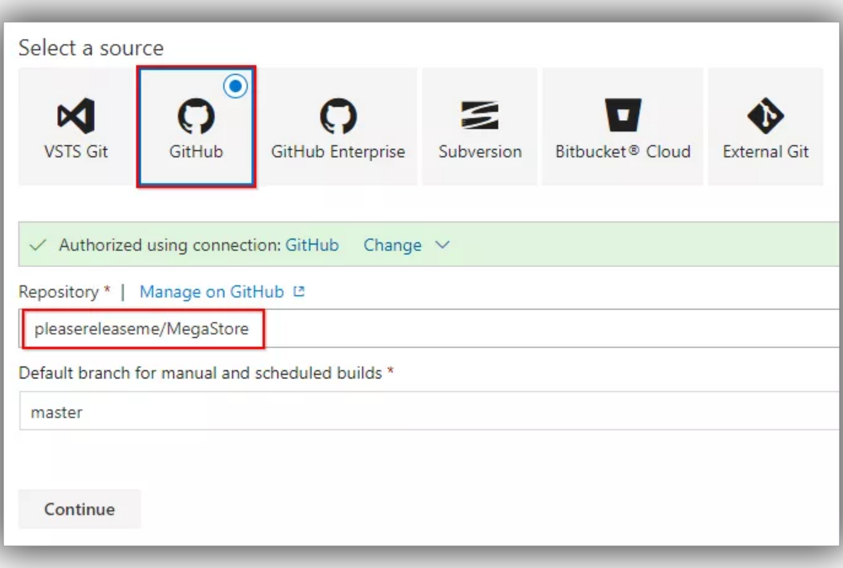
Create Azure SQL Databases for Different Release Pipeline Environments

I'll be creating a release pipeline consisting of DAT and PRD environments. I explain more about these below but to support these environments you'll need to create two new databases—MegaStoreDat and MegaStorePrd. You can do this either through the Azure portal or through SQL Server Management Studio, however be aware that if you use SSMS you'll end up on the standard pricing tier rather than the cheaper basic tier. Either way, you then use SQL Server Management Studio to create dbo.Sale and set up security as described above, ensuring that you create different logins for the different environments.

Create a Build in VSTS

Once everything is working locally the next step is to switch over to VSTS and create a build. I'm assuming that you've cloned my GitHub repo to your own GitHub account however if you are doing it another way (your repo is in VSTS for example) you'll need to amend accordingly.

1. Create a new Build definition in VSTS. The first thing you get asked is to select a repository—link to your GitHub account and select the MegaStore repo:



Authorized using connection: GitHub

Repository *

pleasereleaseme/MegaStore

Default branch for manual and scheduled builds *

master

2. When you get asked to **Choose a template** go for the **Empty process** option.
3. Rename the build to something like **MegaStore** and under **Agent queue** select your private build agent.
4. In the **Triggers** tab check **Enable continuous integration**.
5. In the **Options** tab set **Build number format** to **$(Date:yyyyMMdd)$(Rev:.rr)**, or something meaningful to you based on the available options described here.

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

6. In the **Tasks** tab use the + icon to add two **Docker Compose** tasks and a **Publish Build Artifacts** task. *Note that when configuring the tasks below only the required entries and changes to defaults are listed.*
7. Configure the first Docker Compose task as follows:
   a. Display name = Build service images
   b. Action = Build service images
   c. Azure subscription = [name of existing Azure Resource Manager endpoint]
   d. Azure Container Registry = [name of existing Azure Container Registry]
   e. Additional Image Tags = $(Build.BuildNumber)

8. Configure the first Docker Compose task as follows:
   a. Display name = Push service images
   b. Azure subscription = [name of existing Azure Resource Manager endpoint]
   c. Azure Container Registry = [name of existing Azure Container Registry]
   d. Action = Push service images
   e. Additional Image Tags = $(Build.BuildId)

9. Configure the Publish Build Artifacts task as follows:
   a. Display name = Publish k8s config
   b. Path to publish = k8s
   c. Artifact name = k8s-config
   d. Artifact publish location = Visual Studio Team Services/TFS

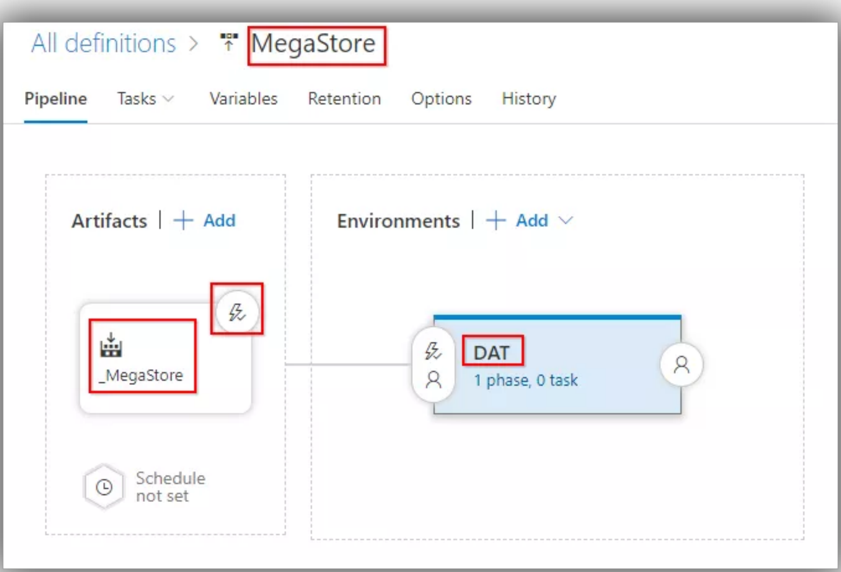
You should now be able to test the build by committing a minor change to the source code. The build should pass and if you look in the **Repositories** section of your Container Registry you should see **megastoreweb** and **megastoresavesalehandler** repositories with newly created images.

Create a DAT Release Environment in VSTS

With the build working it's now time to create the release pipeline, starting with an environment I call **DAT** which is where automated acceptance testing might take place. At this point there is a style choice to be made for creating Kubernetes Secrets and ConfigMaps. They can be configured from files or from literal values. I've gone down the literal values route since the files route needs to specify the namespace and this would require either a separate file for each namespace creating a DRY problem or editing the config files as part of the release pipeline. To me the literal values technique seems cleaner. Either way, as far as I can tell there is no way to update a Secret or ConfigMap via a VSTS **Deploy to Kubernetes** task as it's a two step process and the task can't handle this. The workaround is a task to delete the Secret or ConfigMap and then a task to create it. You'll see that I've also chosen to explicitly create the **image pull secret**. This is partly because of a bug in the Deploy to Kubernetes task however it also

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 avoids having to repeat a lot of the Secrets configuration in Deploy to Kubernetes tasks that deploy service or deployment configurations.

1. Create a new release definition in VSTS, electing to start with an empty process and rename it **MegaStore**.
2. In the **Pipeline** tab click on **Add artifact** and link the build that was just created which in turn makes the **k8s-config** artifact from step 9 above available in the release.
3. Click on the lightning bolt to enable the **Continuous deployment trigger**.
4. Still in the **Pipeline** tab rename **Environment 1** to **DAT**, with the overall changes resulting in something like this:



5. In the **Tasks** tab click on **Agent phase** and under **Agent queue** select your private build agent.
6. In the **Variables** tab create the following variables with **Release** Scope:
   a. AcrAuthenticationSecretName = prmcrauth (or the name you are using for **imagePullSecrets** in the Kubernetes config files)
   b. AcrName = [unique name of your Azure Container Registry, eg mine is **prmcr**]
   c. AcrPassword = [**password** of your Azure Container Registry from **Settings** > **Access keys**], use the padlock to make it a secret

7. In the **Variables** tab create the following variables with **DAT** Scope:
   a. DatDbConn = Server=tcp:megastore.database.windows.net,1433;Initial Catalog=MegaStoreDat;Persist Security Info=False;User ID=sales_user;Password=mystrongpwd;MultipleActiveResultSets=False;Encrypt=True;TrustServerCertificate=Fa Timeout=30; (you will need to alter this connection string for your own Azure SQL server and database)
   b. DatEnvironment = dat (ie in lower case)

8. In the **Tasks** tab add 15 **Deploy to Kubernetes** tasks and disable all but the first one so the release can be tested after each task is configured. *Note that when configuring the tasks below only the required entries and changes to defaults are listed*.
9. Configure the first Deploy to Kubernetes task as follows:
   a. Display name = Delete image pull secret
   b. Kubernetes Service Connection = [name of Kubernetes Service Connection endpoint]
   c. Namespace = $(DatEnvironment)
   d. Command = delete
   e. Arguments = secret $(AcrAuthenticationSecret)
   f. Control Options > Continue on error = checked

10. Configure the second Deploy to Kubernetes task as follows:
   a. Display name = Create image pull secret
   b. Kubernetes Service Connection = [name of Kubernetes Service Connection endpoint]
   c. Namespace = $(DatEnvironment)
   d. Command = create
   e. Arguments = secret docker-registry $(AcrAuthenticationSecretName) --namespace=$(DatEnvironment) --docker-server=$(AcrName).azurecr.io --docker-username=$(AcrName) --docker-password=$(AcrPassword) --docker-email=fred@bloggs.com (note that the email address can be anything you like)

11. Configure the third Deploy to Kubernetes task as follows:
   a. Display name = Delete ASPNETCORE_ENVIRONMENT config map
   b. Kubernetes Service Connection = [name of Kubernetes Service Connection endpoint]
   c. Namespace = $(DatEnvironment)
   d. Command = delete
   e. Arguments = configmap aspnetcore.env
   f. Control Options > Continue on error = checked

12. Configure the fourth Deploy to Kubernetes task as follows:
   a. Display name = Create ASPNETCORE_ENVIRONMENT config map
   b. Kubernetes Service Connection = [name of Kubernetes Service Connection endpoint]
   c. Namespace = $(DatEnvironment)
   d. Command = create
   e. Arguments = configmap aspnetcore.env --from-literal=ASPNETCORE_ENVIRONMENT=$(DatEnvironment)

13. Configure the fifth Deploy to Kubernetes task as follows:
   a. Display name = Delete DB_CONNECTION_STRING secret
   b. Kubernetes Service Connection = [name of Kubernetes Service Connection endpoint]
   c. Namespace = $(DatEnvironment)

   d. Command = delete
   e. Arguments = secret db.connection
   f. Control Options > Continue on error = checked

14. Configure the sixth Deploy to Kubernetes task as follows:
   a. Display name = Create DB_CONNECTION_STRING secret
   b. Kubernetes Service Connection = [name of Kubernetes Service Connection endpoint]
   c. Namespace = $(DatEnvironment)
   d. Command = create
   e. Arguments = secret generic db.connection --from-literal=DB_CONNECTION_STRING="$(DatDbConn)"

15. Configure the seventh Deploy to Kubernetes task as follows:
   a. Display name = Delete MESSAGE_QUEUE_URL config map
   b. Kubernetes Service Connection = [name of Kubernetes Service Connection endpoint]
   c. Namespace = $(DatEnvironment)
   d. Command = delete
   e. Arguments = configmap message.queue
   f. Control Options > Continue on error = checked

16. Configure the eighth Deploy to Kubernetes task as follows:
   a. Display name = Create MESSAGE_QUEUE_URL config map
   b. Kubernetes Service Connection = [name of Kubernetes Service Connection endpoint]
   c. Namespace = $(DatEnvironment)
   d. Command = create
   e. Arguments = configmap message.queue --from-literal=MESSAGE_QUEUE_URL=nats://message-queue-service.$(DatEnvironment):4222

17. Configure the ninth Deploy to Kubernetes task as follows:
   a. Display name = Create message-queue service
   b. Kubernetes Service Connection = [name of Kubernetes Service Connection endpoint]
   c. Namespace = $(DatEnvironment)
   d. Command = apply
   e. Use Configuration files = checked
   f. Configuration File = $(System.DefaultWorkingDirectory)/_MegaStore/k8s-config/message-queue-service.yaml

18. Configure the tenth Deploy to Kubernetes task as follows:
   a. Display name = Create megastore-web service
   b. Kubernetes Service Connection = [name of Kubernetes Service Connection endpoint]
   c. Namespace = $(DatEnvironment)

   d. Command = apply
   e. Use Configuration files = checked
   f. Configuration File = $(System.DefaultWorkingDirectory)/_MegaStore/k8s-config/megastore-web-service.yaml

19. Configure the eleventh Deploy to Kubernetes task as follows:
   a. Display name = Create message-queue deployment
   b. Kubernetes Service Connection = [name of Kubernetes Service Connection endpoint]
   c. Namespace = $(DatEnvironment)
   d. Command = apply
   e. Use Configuration files = checked
   f. Configuration File = $(System.DefaultWorkingDirectory)/_MegaStore/k8s-config/message-queue-deployment.yaml

20. Configure the twelfth Deploy to Kubernetes task as follows:
   a. Display name = Create megastore-web deployment
   b. Kubernetes Service Connection = [name of Kubernetes Service Connection endpoint]
   c. Namespace = $(DatEnvironment)
   d. Command = apply
   e. Use Configuration files = checked
   f. Configuration File = $(System.DefaultWorkingDirectory)/_MegaStore/k8s-config/message-queue-deployment.yaml

21. Configure the thirteenth Deploy to Kubernetes task as follows:
   a. Display name = Update megastore-web with latest image
   b. Kubernetes Service Connection = [name of Kubernetes Service Connection endpoint]
   c. Namespace = $(DatEnvironment)
   d. Command = set
   e. Arguments = image deployment/megastore-web-deployment megastoreweb=$(AcrName).azurecr.io/megastoreweb:$(Build.BuildNumber)

22. Configure the fourteenth Deploy to Kubernetes task as follows:
   a. Display name = Create megastore-savesalehandler deployment
   b. Kubernetes Service Connection = [name of Kubernetes Service Connection endpoint]
   c. Namespace = $(DatEnvironment)
   d. Command = apply
   e. Use Configuration files = checked
   f. Configuration File = $(System.DefaultWorkingDirectory)/_MegaStore/k8s-config/megastore-savesalehandler-deployment.yaml

23. Configure the fifthteenth Deploy to Kubernetes task as follows:
    a. Display name = Update megastore-savesalehandler with latest image
    b. Kubernetes Service Connection = [name of Kubernetes Service Connection endpoint]
    c. Namespace = $(DatEnvironment)
    d. Command = set
    e. Arguments = image deployment/megastore-savesalehandler-deployment megastoresavesalehandler=$(AcrName).azurecr.io/megastoresavesalehandler:$(Build.BuildNumber)

That's a heck of a lot of configuration, so what exactly have we built?

The first eight tasks deal with the configuration that support the services and deployments:

- The **image pull secret** stores the credentials to the Azure Container Registry so that deployments that need to pull images from the ACR can authenticate.
- The **ASPNETCORE_ENVIRONMENT config map** sets the environment for ASP.NET Core. I don't do anything with this but it could be handy for troubleshooting purposes.
- The **DB_CONNECTION_STRING secret** stores the connection string to the Azure SQL database and is used by the **megastore-savesalehandler-deployment.yaml** configuration.
- The **MESSAGE_QUEUE_URL config map** stores the URL to the NATS message queue and is used by the **megastore-web-deployment.yaml** and **megastore-savesalehandler-deployment.yaml** configurations.

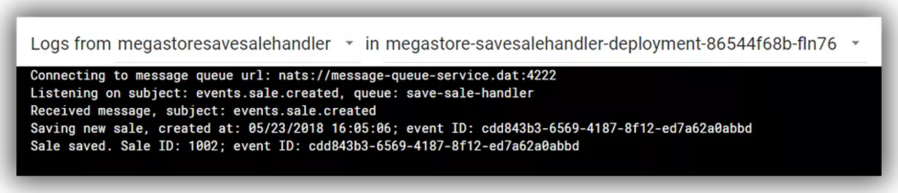
As mentioned above, a limitation of the VSTS **Deploy to Kubernetes** task means that in order to be able to update Secrets and ConfigMaps they need to be deleted first and then created again. This does mean that an exception is thrown the first time a delete task is run however the **Continue on error** option ensures that the release doesn't fail.

The remaining seven tasks deal with the deployment and configuration of the components (other than the Azure SQL database) that make up the MegaStore application:

- The NATS message queue requires a service so other components can talk to it and the deployment that specifies the specification for the image.
- The MegaStore.Web front end requires a service so that it is exposed to the outside world and the deployment that specifies the specification for the image.
- MegaStore.SaveSalehandler monitoring component only needs the deployment that specifies the specification for the image as nothing connects to it directly.

If everything has been configured correctly then triggering a release should result in a **megastore-web-service** being created. You can check the deployment was successful by executing **kubectl get services --namespace=dat** to get the external IP address of the **LoadBalancer** which you can paste in to a browser to confirm that the ASP.NET Core website is running. On the backend, you can use SQL Server Management Studio to connect to the database and confirm that records are being created in **dbo.Sale**.

If you are running in to problems, you can run the Kubernetes Dashboard to find out what is failing. Typically it's deployments that fail, and navigating to **Workloads** > **Deployments** can highlight the failing deployment. You can find out what the error is from the **New Replica Set** panel by clicking on the **Logs** icon which brings up a new browser tab with a command line style output of the error. If there is no error it displays any **Console.WiteLine** output. Very neat:


Logs from megastoresavesalehandler
in megastore-savesalehandler-deployment-86544f68b-fln76

```
Connecting to message queue url: nats://message-queue-service.dat:4222
Listening on subject: events.sale.created, queue: save-sale-handler
Received message, subject: events.sale.created
Saving new sale, created at: 05/23/2018 16:05:06; event ID: cdd843b3-6569-4187-8f12-ed7a62a0abbd
Sale saved. Sale ID: 1002; event ID: cdd843b3-6569-4187-8f12-ed7a62a0abbd
```

Create a PRD Release Environment in VSTS

With a DAT environment created we can now create other environments on the route to production. This could be whatever else is needed to test the application, however here I'm just going to create a production environment I'll call PRD. I described this process in my previous post so here I'll just list the high level process:

1. Clone the DAT environment and rename it PRD.
2. In the **Variables** tab rename the cloned DatDbConn and DatEnvironment variables (the ones with PRD scope) to **PrdDbConn** and **PrdEnvironment** and change their values accordingly.
3. In the **Tasks** tab visit each task and change all references of $(DatDbConn) and $(DatEnvironment) to $(PrdDbConn) and $(PrdEnvironment). All **Namespace** fields will need changing and many of the tasks with use the **Arguments** fields will need attention.
4. Trigger a build and check the deployment was successful by executing **kubectl get services --namespace=prd** to get the external IP address of the **LoadBalancer** which you can paste in to a browser to confirm that the ASP.NET Core website is running.

Wrapping Up

Although the final result is a CI/CD pipeline that certainly works there are more tasks than I'm happy with due to the need to delete and then recreate

Secrets and ConfigMaps and this also adds quite a bit of overhead to the time it takes to deploy to an environment. There's bound to be a more elegant way of doing this that either exists now and I just don't know about it or that will exist in the future. Do post in the comments if you have thoughts.

Although I'm three posts in I've barely scratched the surface of the different topics that I could cover, so plenty more to come in this series. Next time it will probably be around health and / or monitoring.

Cheers—Graham

Share this:

Share 0 Share Tweet G+ 5

Save Email Print

Posted in **Continuous Delivery with Containers**

Tags: Kubernetes NATS Visual Studio Team Services VSTS

0 Comments **pleasereleaseme** 1 **Login**

Recommend Tweet Share **Sort by Best**

Start the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

Be the first to comment.

ALSO ON **PLEASERELEASEME**

Deploy a Dockerized ASP.NET Core Application to

5 comments • 9 months ago

Graham Smith — Hi JohnThe prmcr.azurecr.io bit is the FQDN of the Azure Container

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

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

Continuous Delivery with Containers – Azure CLI

6 comments • 2 years ago

Graham Smith — Thanks for the update!

Subscribe **Add Disqus to your siteAdd DisqusAdd**

Disqus' Privacy PolicyPrivacy PolicyPrivacy

Powered by WordPress, Theme by Andrew Dyer