Azure Bootcamp 2018

Contents

Prereqs

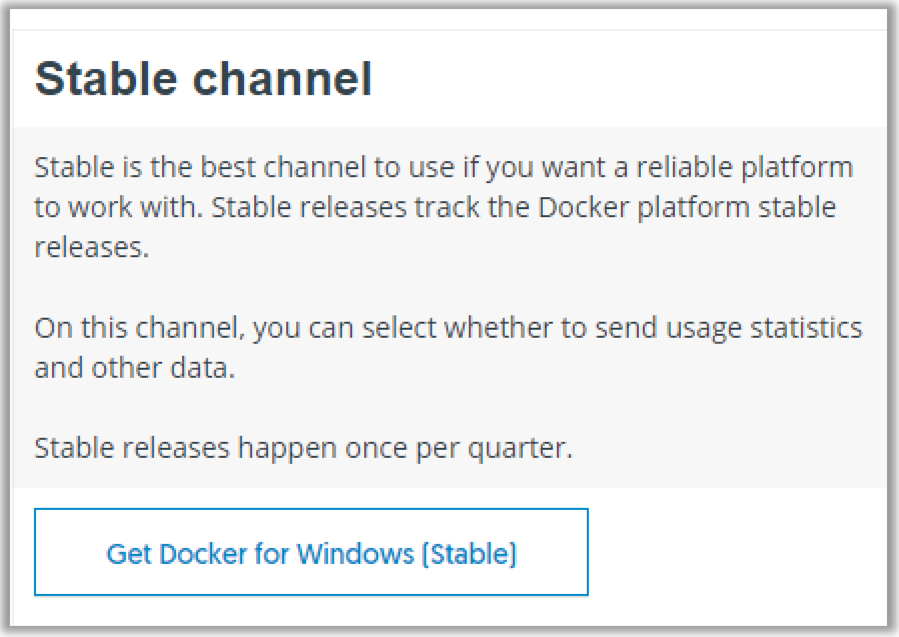
Don’t worry if you don’t have all of these yet!

- Docker for Windows
- SQL Server 2017 Docker image.
- Azure CLI for PowerShell.
- SQL Server Management Studio.
- Postman.

Getting started

You can do this while Bernard is talking if necessary.

- $ git clone https://github.com/bernardoleary/Azure-Bootcamp-2018.git
- Install Docker for Windows (D4W) – shouldn’t take too long: https://docs.docker.com/docker-for-windows/install/



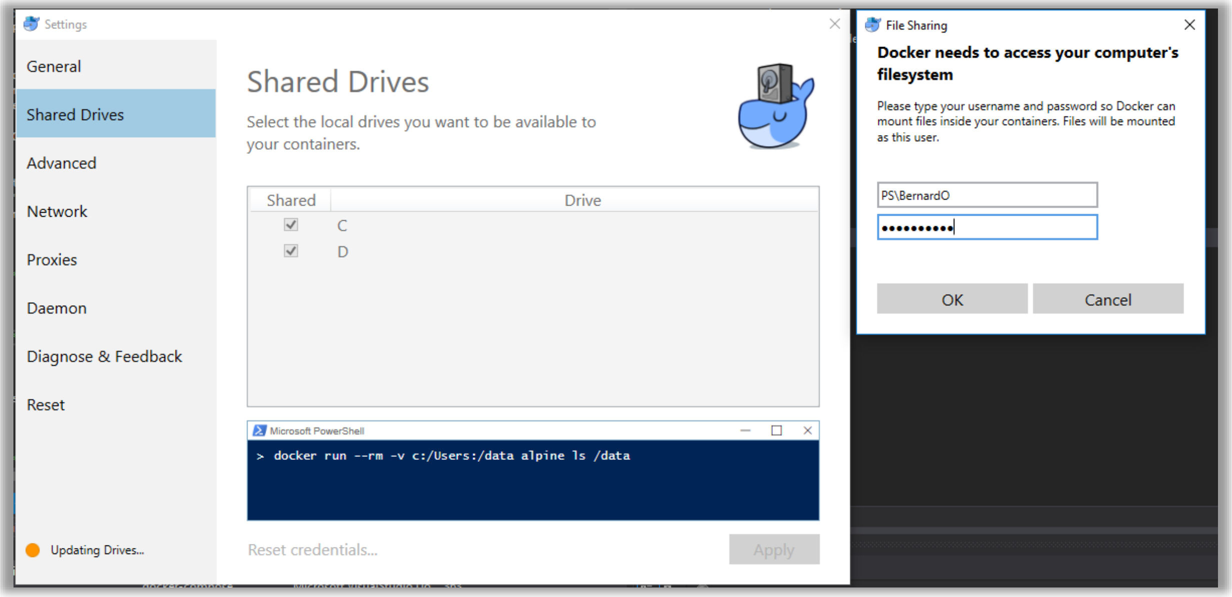
Stable channel

Stable is the best channel to use if you want a reliable platform to work with. Stable releases track the Docker platform stable releases.

On this channel, you can select whether to send usage statistics and other data.

Stable releases happen once per quarter.

Make sure that you have enabled shared access across your drives:



Shared Drives

Select the local drives you want to be available to your containers.

Shared	Drive
☑	C
☑	D

```
> docker run --rm -v c:/Users:/data alpine ls /data
```

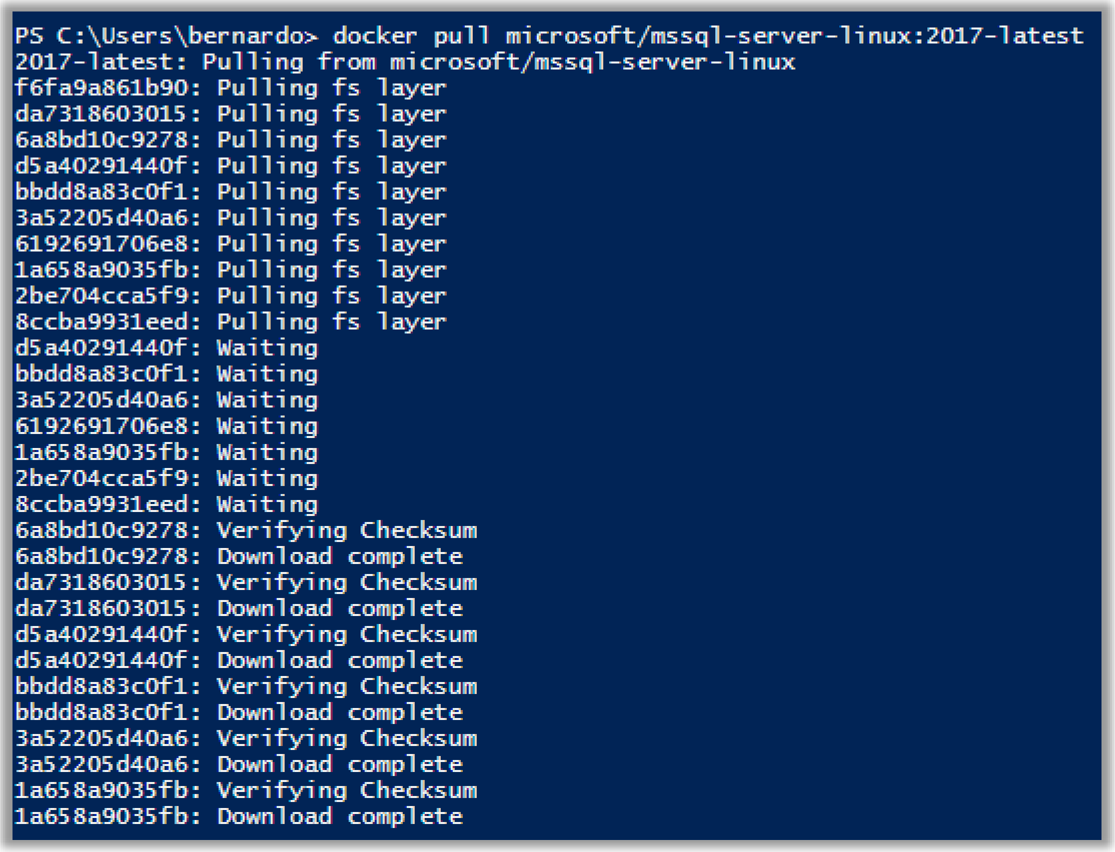
Docker needs to access your computer's filesystem

Please type your username and password so Docker can mount files inside your containers. Files will be mounted as this user.

PS\BernardO

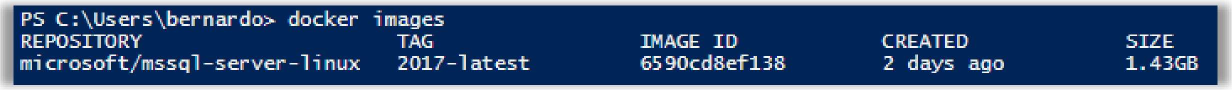
••••••••••

- Get the SQL Server 2017 image – might take a little while:
  $ docker pull microsoft/mssql-server-linux:2017-latest



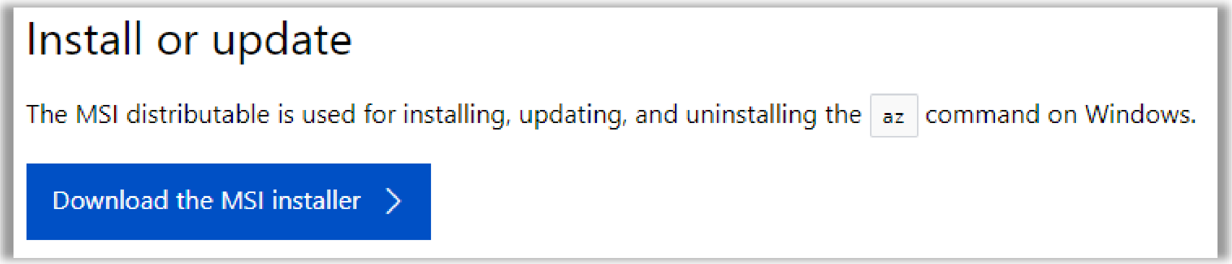
```
PS C:\Users\bernardo> docker pull microsoft/mssql-server-linux:2017-latest
2017-latest: Pulling from microsoft/mssql-server-linux
f6fa9a861b90: Pulling fs layer
da7318603015: Pulling fs layer
6a8bd10c9278: Pulling fs layer
d5a40291440f: Pulling fs layer
bbdd8a83c0f1: Pulling fs layer
3a52205d40a6: Pulling fs layer
6192691706e8: Pulling fs layer
1a658a9035fb: Pulling fs layer
2be704cca5f9: Pulling fs layer
8ccba9931eed: Pulling fs layer
d5a40291440f: Waiting
bbdd8a83c0f1: Waiting
3a52205d40a6: Waiting
6192691706e8: Waiting
1a658a9035fb: Waiting
2be704cca5f9: Waiting
8ccba9931eed: Waiting
6a8bd10c9278: Verifying Checksum
6a8bd10c9278: Download complete
da7318603015: Verifying Checksum
da7318603015: Download complete
d5a40291440f: Verifying Checksum
d5a40291440f: Download complete
bbdd8a83c0f1: Verifying Checksum
bbdd8a83c0f1: Download complete
3a52205d40a6: Verifying Checksum
3a52205d40a6: Download complete
1a658a9035fb: Verifying Checksum
1a658a9035fb: Download complete
```

- Confirm download:
  $ docker images



```
PS C:\Users\bernardo> docker images
REPOSITORY                     TAG                 IMAGE ID            CREATED             SIZE
microsoft/mssql-server-linux   2017-latest         6590cd8ef138        2 days ago          1.43GB
```

- Install the Azure CLI: https://docs.microsoft.com/en-us/cli/azure/install-azure-cli-windows?view=azure-cli-latest



Install or update

The MSI distributable is used for installing, updating, and uninstalling the `az` command on Windows.

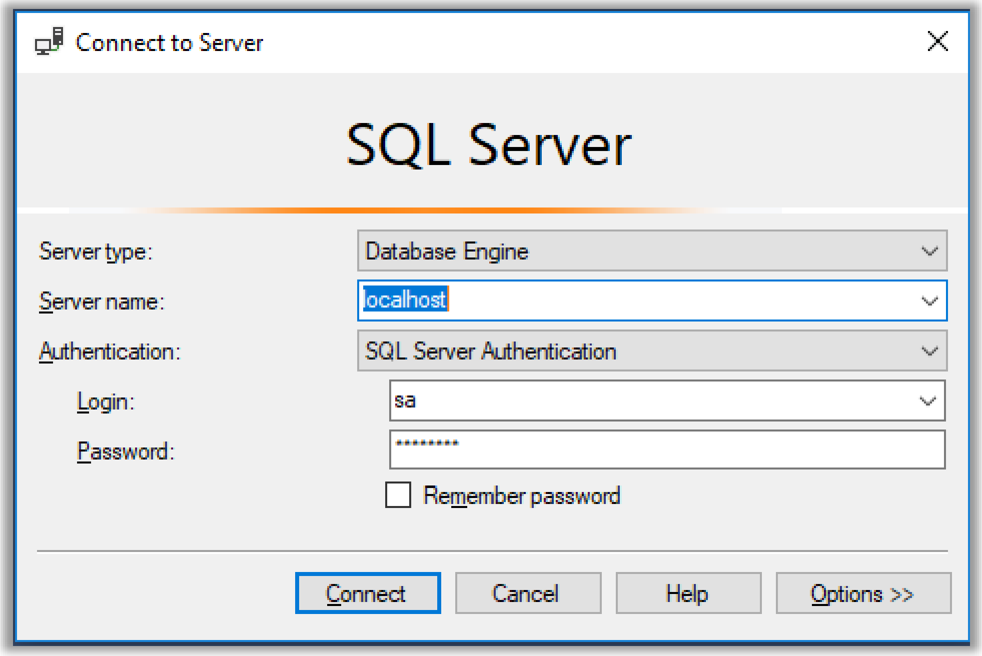
- Make sure you have Visual Studio Tool for Docker installed: https://docs.microsoft.com/en-us/dotnet/standard/containerized-lifecycle-architecture/design-develop-containerized-apps/visual-studio-tools-for-docker
- Make sure you have SSMS installed: https://docs.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-2017
- Make sure you have Postman (or similar) installed: https://www.getpostman.com/

Workshop 1 – Inventory “microservice” on Docker

We’re going to build an extremely simple microservice (hence the quotation marks) using Docker for Windows, Docker Compose and SQL Server 2017 on Docker. Then we’re going to populate it using Postman.

Run a SQL Server container

1. Run the SQL image – create a container (change the port if you have a default SQL Server instance running on your machine already):
   ```
   $ docker run -e "ACCEPT_EULA=Y" -e "MSSQL_SA_PASSWORD=P@ssw0rd" -p 1433:1433 --name sql1 -d microsoft/mssql-server-linux:2017-latest
   ```
2. Verify the container is running:
   ```
   $ docker ps
   $ docker container ls
   ```
3. Log in to the DB server:



Connect to Server

SQL Server

Server type: Database Engine

Server name: localhost

Authentication: SQL Server Authentication

Login: sa

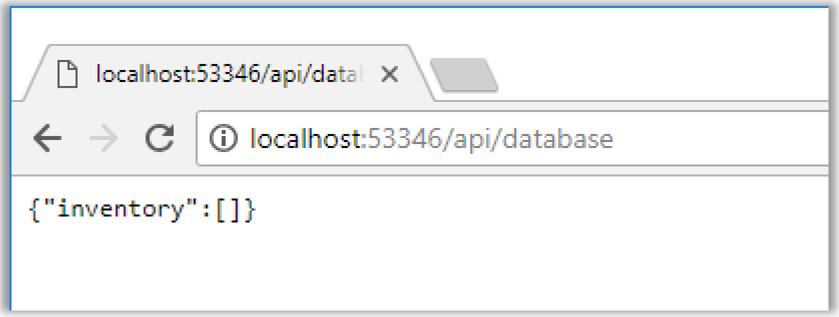
Password: *********

□ Remember password

4. Run the SQL script to create the InventoryDB:
   https://gist.github.com/bernardoleary/faf1e515d40f38db7fcf2aeb29bb4a3b
   Populate the InventoryDB with some stuff manually if you like.

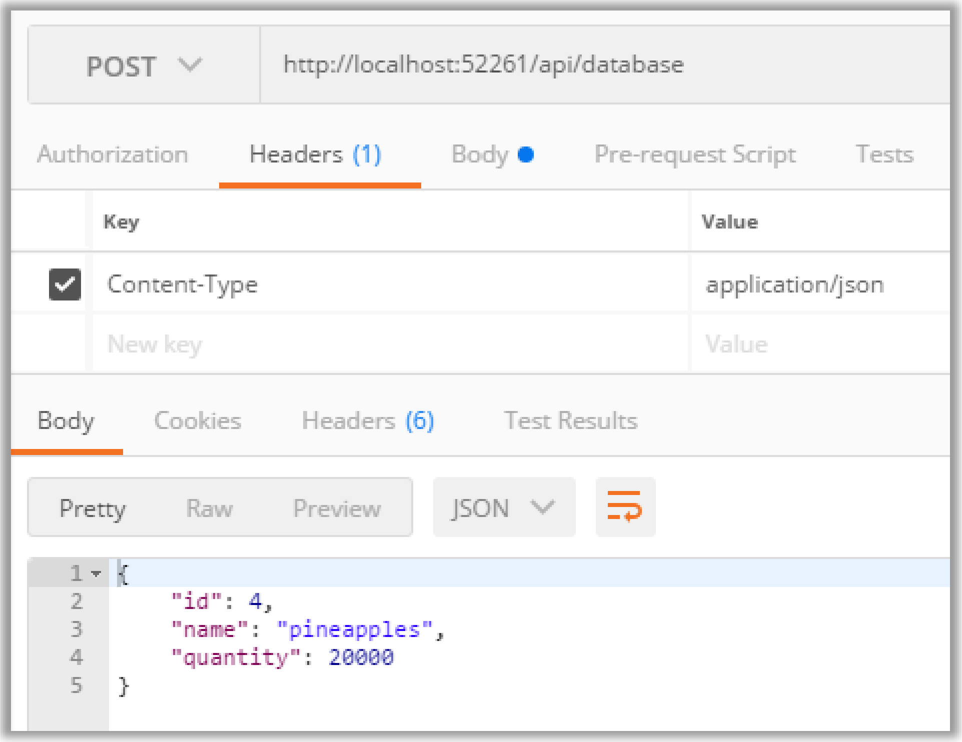
Get an API going

5. Get the code for the demo solution:
   ```
   $ git clone
   ```
6. Take a look at Startup.cs – especially if you have not used dotnet core much in the past, note the baked-in IoC management (IServiceCollection). Also note the database connection details.
7. Run the solution and browse to the DB controller:



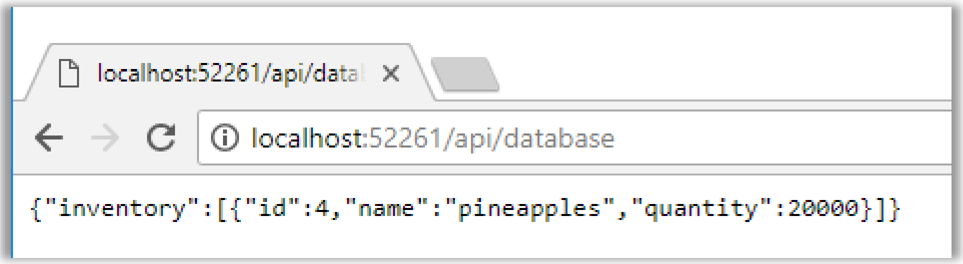
```
{"inventory":[]}
```

8. Now we can populate the DB using Postman:
   {"id":4,"name":"pineapples","quantity":20000}
   http://localhost:53346/api/database
   Content-Type application/json



	Key	Value
☑	Content-Type	application/json
	New key	Value

```
{
    "id": 4,
    "name": "pineapples",
    "quantity": 20000
}
```

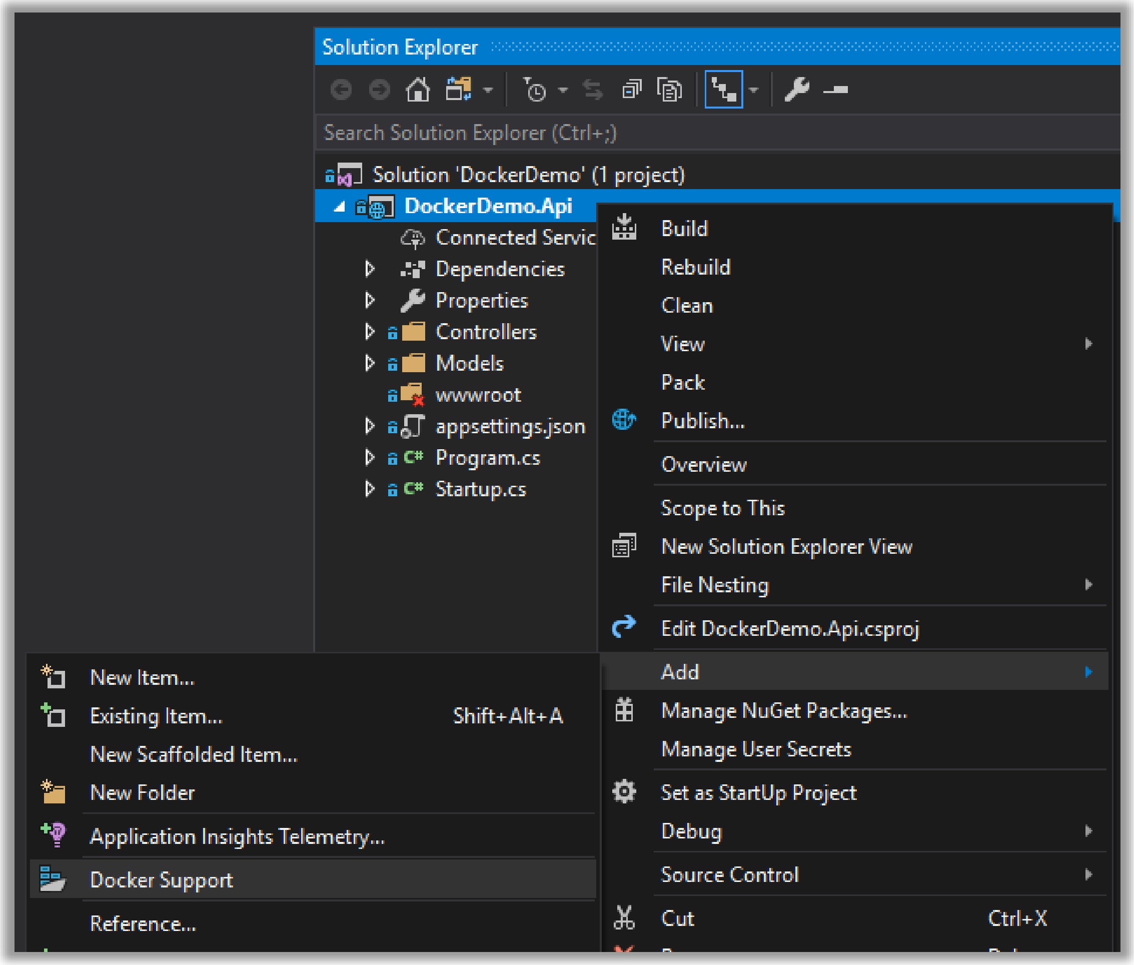


```
{"inventory":[{"id":4,"name":"pineapples","quantity":20000}]}
```

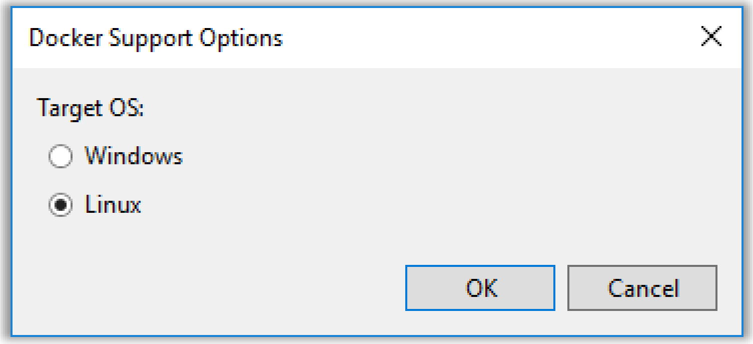
9. Note that this solution is not running as a container – it is on IIS Express – but the database is on a container.

Add Docker Support

10. Now add "Docker Support" to the dotnet core project:



Target Linux (because our SQL Server container is running on Linux also):



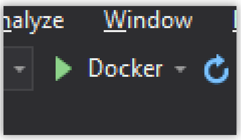
Docker Support Options

Target OS:

○ Windows

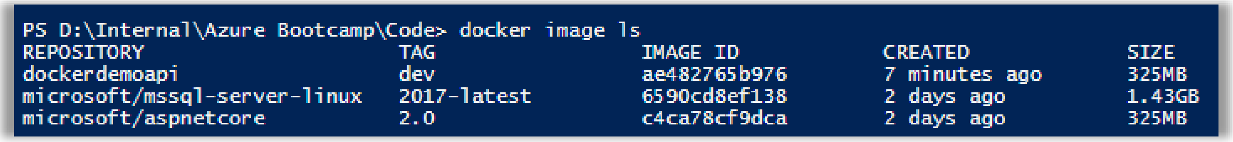
● Linux

This will add another project to your solution called “docker-compose”. Note that your debug options have changed to “Docker” only (“IISExpress” is gone). Also a file called “Dockerfile” has been added to your dotnet core project.



11. Run the Docker-ised solution in debug mode.
    What happens when you try to reach http://localhost:<port>/api/values?
    What happens when you try to reach http://localhost: <port>/api/database?
12. Go to your PowerShell prompt and run:

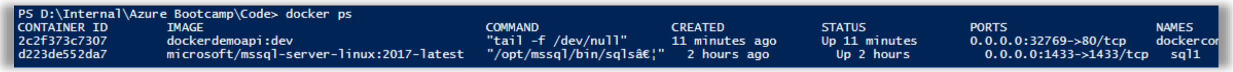
$ docker image ls
This will list all images – note that there are two new images there now – one for your application and one for microsoft/aspnetcore:



```
PS D:\Internal\Azure Bootcamp\Code> docker image ls
REPOSITORY                      TAG                 IMAGE ID            CREATED             SIZE
dockerdemoapi                   dev                 ae482765b976        7 minutes ago       325MB
microsoft/mssql-server-linux    2017-latest         6590cd8ef138        2 days ago          1.43GB
microsoft/aspnetcore            2.0                 c4ca78cf9dca        2 days ago          325MB
```

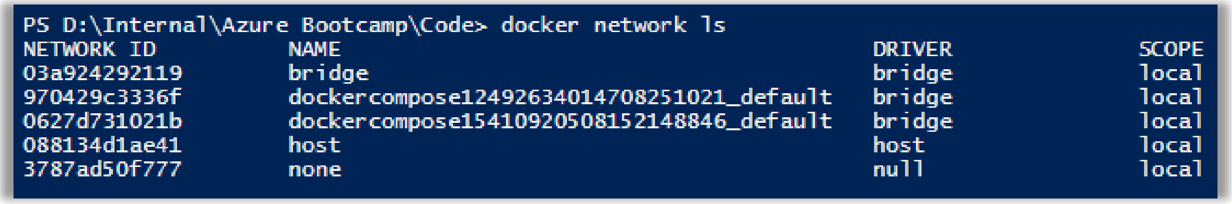
Docker has downloaded the microsoft/aspnetcore image because it is required to run our dotnet core web application – this is specified in the Dockerfile (take a look).

13. Now list your running containers:
    $ docker image ls
    You should see there are two containers running – one is out SQL DB, the other is our dotnet core web application:



```
PS D:\Internal\Azure Bootcamp\Code> docker ps
CONTAINER ID        IMAGE                                      COMMAND                  CREATED             STATUS              PORTS                    NAMES
2c2f373c7307        dockerdemoapi:dev                          "tail -f /dev/null"      11 minutes ago      Up 11 minutes       0.0.0.0:32769->80/tcp    dockercor
d223de552da7        microsoft/mssql-server-linux:2017-latest   "/opt/mssql/bin/sqlsâ€¦"   2 hours ago         Up 2 hours          0.0.0.0:1433->1433/tcp   sql1
```

Why can the web-app not see the DB? Remember that Docker is like a miniature datacentre running on your PC – complete with networks. You can inspect the networks and see what containers are running on them:
$ docker network ls

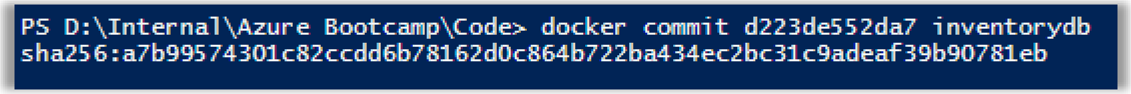


```
PS D:\Internal\Azure Bootcamp\Code> docker network ls
NETWORK ID          NAME                                       DRIVER              SCOPE
03a924292119        bridge                                     bridge              local
970429c3336f        dockercompose1249263401470825l021_default  bridge              local
0627d731021b        dockercompose15410920508152148846_default  bridge              local
088134d1ae41        host                                       host                local
3787ad50f777        none                                       null                local
```

$ docker network inspect <network id>
Unless it is told to, Docker will not put containers on the same network – hence why our containers aren't able to see each other.

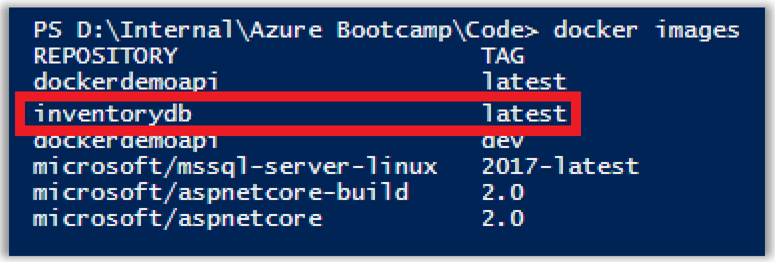
Take a copy of the DB and reset

14. Because Docker images are stateless, when we create a container from one it will spawn from scratch (a blank DB server) – so to avoid having to recreate the DB, we take a copy of the running container and commit it as an image – like this (get the container ID by running a "$ docker ps"):



```
PS D:\Internal\Azure Bootcamp\Code> docker commit d223de552da7 inventorydb
sha256:a7b99574301c82ccdd6b78162d0c864b722ba434ec2bc31c9adeaf39b90781eb
```

$ docker image ls



```
PS D:\Internal\Azure Bootcamp\Code> docker images
REPOSITORY                      TAG
dockerdemoapi                   latest
inventorydb                     latest
dockerdemoapi                   dev
microsoft/mssql-server-linux    2017-latest
microsoft/aspnetcore-build      2.0
microsoft/aspnetcore            2.0
```

15. Stop debugging and clear out all containers:
    $ docker stop $(docker ps -a -q)

$ docker rm $(docker ps -a -q)
Check that no containers are running:
$ docker ps

Add docker-compose support

16. Open the file named docker-compose.yml. Note that only our web-app's container is listed. We need to start the inventory container at the same time using docker-compose so that they containers are on the same network. Add the highlighted lines to the docker file:

```
version: '3'

services:
  dockerdemo.api:
    image: dockerdemoapi
    build:
      context: .
      dockerfile: DockerDemo.Api/Dockerfile
  dockerdemodb:
    image: inventorydb
```

17. Open Startup.cs and change the following line of code:

```
Environment.GetEnvironmentVariable("SQLSERVER_HOST") ?? " dockerdemodb";
```

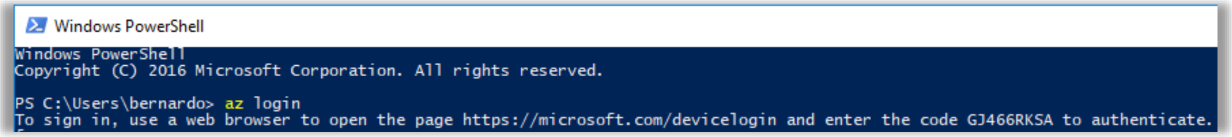
18. Run the solution in debug mode again and try to reach the "/api/database" endpoint. Run a "$ docker ps" to see that you have your two containers running. The API container can now look up the hostname dockerdemodb and get back that container's IP address. Same goes for any other container we run as part of this solution using docker-compose.

Workshop 2 – Push the Inventory microservice to AKS

We're going to make a Docker Hub account, push our Docker images to Docker Hub and finally we'll launch them to managed Kubernetes on Azure (AKS).

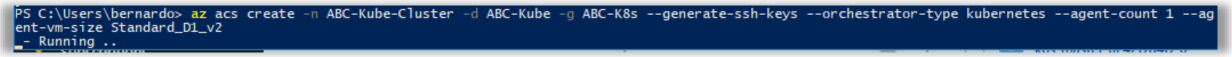
Deploy an AKS cluster

1. Open PowerShell and login to Azure:
   $ az login



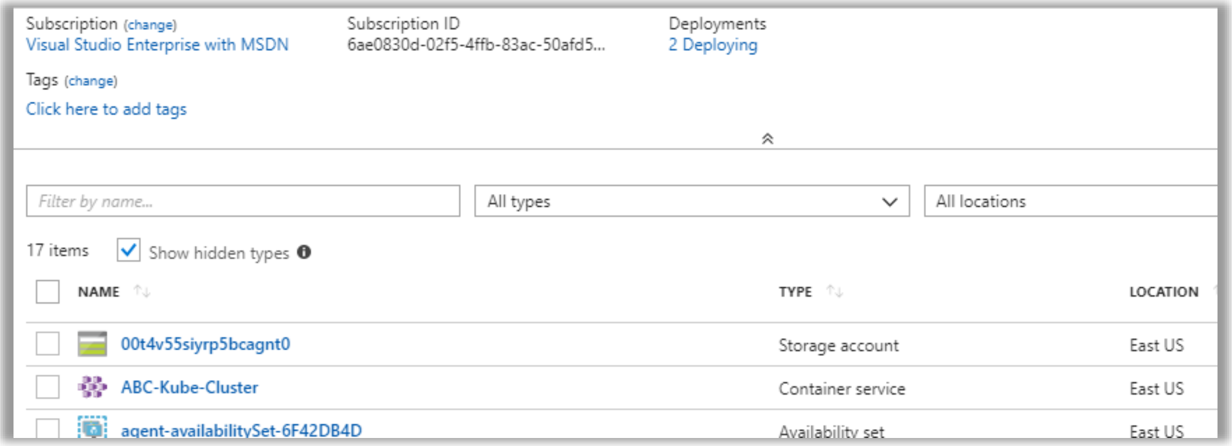
```
Windows PowerShell
Copyright (C) 2016 Microsoft Corporation. All rights reserved.

PS C:\Users\bernardo> az login
To sign in, use a web browser to open the page https://microsoft.com/devicelogin and enter the code GJ466RKSA to authenticate.
```

   Set our subscription ID:
   $ az account set --subscription "<subscription id>"
2. Create a new Resource Group for this workshop, which we'll put the K8s (K8s is short for Kubernetes) cluster on – note, the Resource Group must be in eastus or another region that support the AKS preview:
   $ az group create -l eastus -n ABC- K8s
3. Create our K8s cluster – this will take about 10-to-15 minutes:
   $ az acs create -n ABC-Kube-Cluster -d ABC-Kube -g ABC-K8s --generate-ssh-keys --orchestrator-type kubernetes --agent-count 1 --agent-vm-size Standard_D1_v2



```
PS C:\Users\bernardo> az acs create -n ABC-Kube-Cluster -d ABC-Kube -g ABC-K8s --generate-ssh-keys --orchestrator-type kubernetes --agent-count 1 --ag
ent-vm-size Standard_D1_v2
- Running ..
```

Once you see the “Running ..” prompt you should be able to see the cluster deploying on the Azure portal:



Subscription (change)
Visual Studio Enterprise with MSDN

Subscription ID
6ae0830d-02f5-4ffb-83ac-50afd5...

Deployments
2 Deploying

Tags (change)
Click here to add tags

Filter by name... | All types | All locations

17 items ☑ Show hidden types

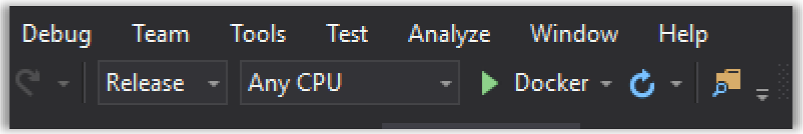
□	NAME	TYPE	LOCATION
□	00t4v55siyrp5bcagnt0	Storage account	East US
□	ABC-Kube-Cluster	Container service	East US
□	agent-availabilitySet-6F42DB4D	Availability set	East US

Set up your Docker Hub account

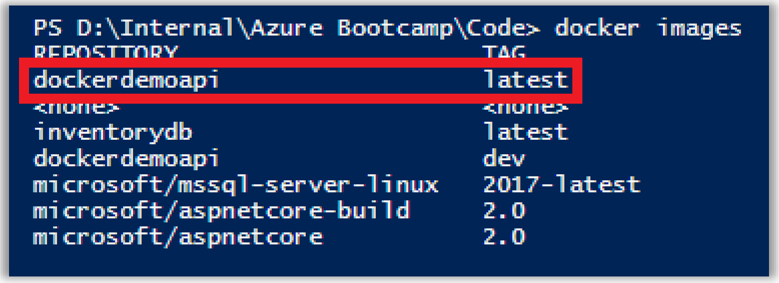
4. Go to https://hub.docker.com and sign-up – too easy!

Make a release build of the microservice API

5. Go back to VS and change your build mode to “Release” and re-run our microservice:



Note that once the build/run is completed that there is a new tag in our dockerdemoapi image tagged “latest”. Previously we only had one image in the repo, with a “dev” tag:



```
PS D:\Internal\Azure Bootcamp\Code> docker images
REPOSITORY                      TAG
dockerdemoapi                   latest
<none>                          <none>
inventorydb                     latest
dockerdemoapi                   dev
microsoft/mssql-server-linux    2017-latest
microsoft/aspnetcore-build      2.0
microsoft/aspnetcore            2.0
```

6. Test that the microservice works by changing our docker-compose.yml and docker-compose.override.yml file subtly to match our new images and running “$ docker-compose up” and/or running the release build in VS:

docker-compose.override.yml

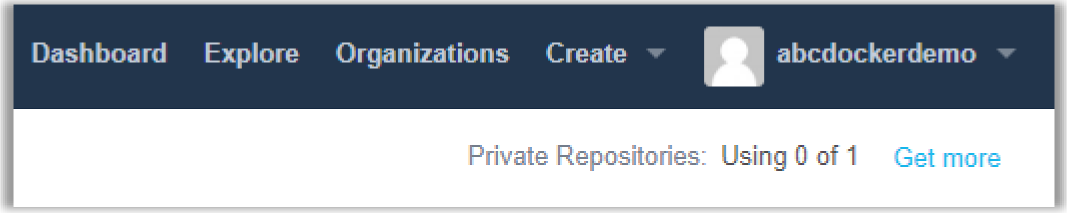
```
version: '3'

services:
  dockerdemoapi:
    environment:
      - ASPNETCORE_ENVIRONMENT=Development
    ports:
             - "80"
```

docker-compose.yml

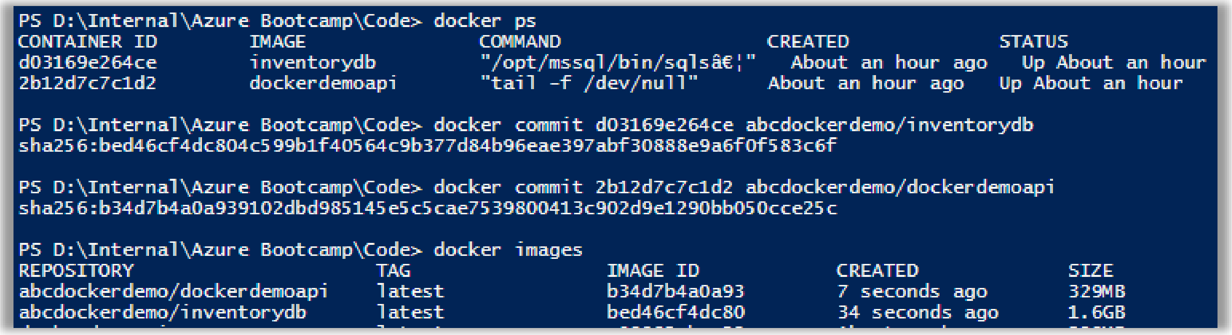
```
version: '3'

services:
  dockerdemoapi:
    image: <your docker hub namespace>/dockerdemoapi
    build:
      context: .
      dockerfile: DockerDemo.Api/Dockerfile
  dockerdemodb:
    image: <your docker hub namespace>/inventorydb
```

7. Using the Docker "ps" and "commit" commands, make copies of your images that are prefixed with the namespace that you have created for your Docker Hub profile.

   Docker Hub namespace – my one is "abcdockerdemo":



Commit command to make images that are prefixed with your Docker Hub namespace:



```
PS D:\Internal\Azure Bootcamp\Code> docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
d03169e264ce        inventorydb         "/opt/mssql/bin/sqlsâ€¦"   About an hour ago   Up About an hour
2b12d7c7c1d2        dockerdemoapi       "tail -f /dev/null"      About an hour ago   Up About an hour

PS D:\Internal\Azure Bootcamp\Code> docker commit d03169e264ce abcdockerdemo/inventorydb
sha256:bed46cf4dc804c599b1f40564c9b377d84b96eae397abf30888e9a6f0f583c6f

PS D:\Internal\Azure Bootcamp\Code> docker commit 2b12d7c7c1d2 abcdockerdemo/dockerdemoapi
sha256:b34d7b4a0a939102dbd985145e5c5cae7539800413c902d9e1290bb050cce25c

PS D:\Internal\Azure Bootcamp\Code> docker images
REPOSITORY                    TAG                 IMAGE ID            CREATED             SIZE
abcdockerdemo/dockerdemoapi   latest              b34d7b4a0a93        7 seconds ago       329MB
abcdockerdemo/inventorydb     latest              bed46cf4dc80        34 seconds ago      1.6GB
```

Push your images to Docker Hub

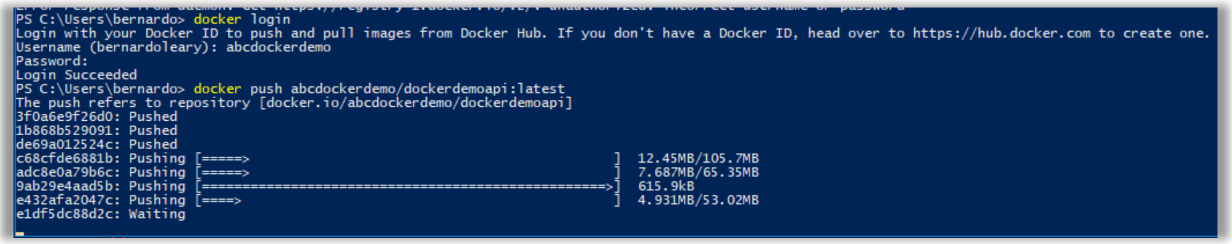
8. Login to Docker:

   $ docker login

9. Push your images to Docker Hub:

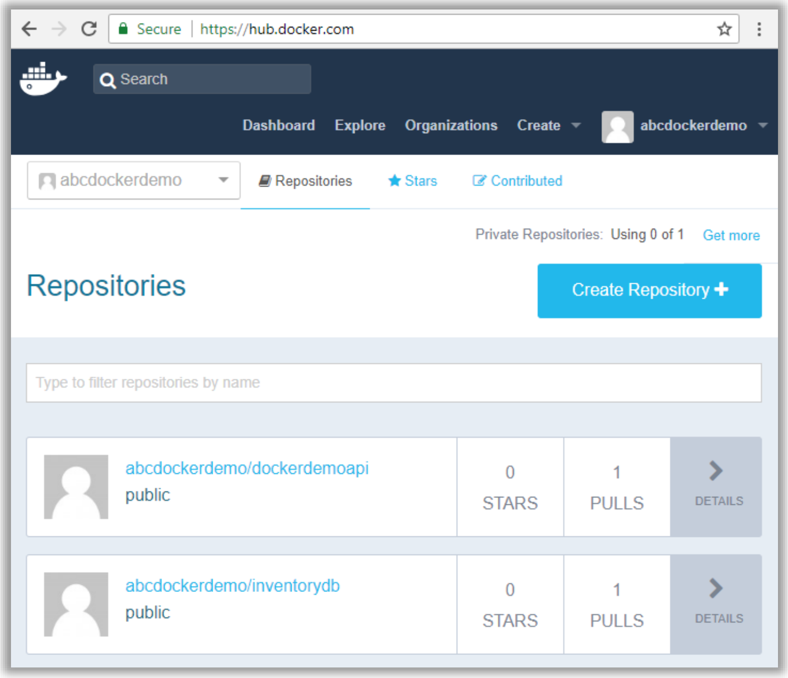
   $ docker push <your docker hub namespace>/dockerdemoapi:latest

   $ docker push <your docker hub namespace>/inventorydb:latest



```
Error response from daemon: Get https://registry-1.docker.io/v2/: unauthorized: incorrect username or password
PS C:\Users\bernardo> docker login
Login with your Docker ID to push and pull images from Docker Hub. If you don't have a Docker ID, head over to https://hub.docker.com to create one.
Username (bernardoleary): abcdockerdemo
Password:
Login Succeeded
PS C:\Users\bernardo> docker push abcdockerdemo/dockerdemoapi:latest
The push refers to repository [docker.io/abcdockerdemo/dockerdemoapi]
3f0a6e9f26d0: Pushed
1b868b529091: Pushed
de69a012524c: Pushed
c68cfde6881b: Pushing [=====>                                             ]  12.45MB/105.7MB
adc8e0a79b6c: Pushing [=====>                                             ]  7.687MB/65.35MB
9ab29e4aad5b: Pushing [==================================================>]  615.9kB
e432afa2047c: Pushing [====>                                              ]  4.931MB/53.02MB
e1df5dc88d2c: Waiting
```

You should be able to see your new repos on Docker Hub when you have finished:


Secure | https://hub.docker.com
Search
Dashboard Explore Organizations Create abcdockerdemo
abcdockerdemo
Repositories
Stars
Contributed
Create Repository
Type to filter repositories by name
DETAILS
DETAILS

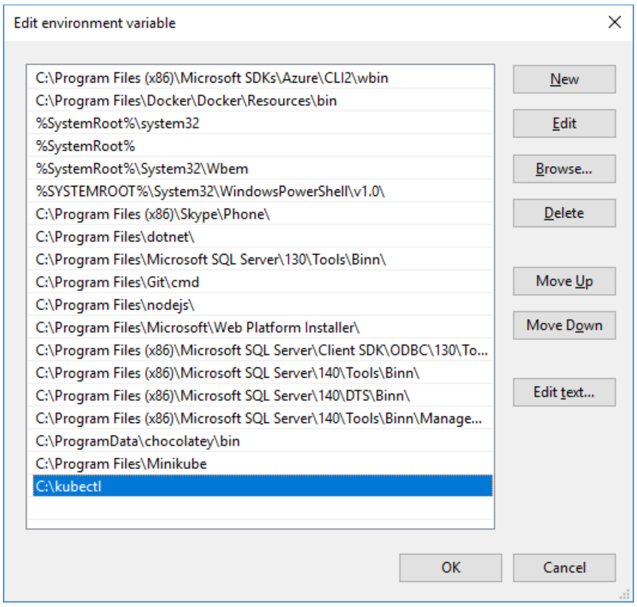
Private Repositories: Using 0 of 1 Get more

Repositories

abcdockerdemo/dockerdemoapi public	0 STARS	1 PULLS	DETAILS
abcdockerdemo/inventorydb public	0 STARS	1 PULLS	DETAILS

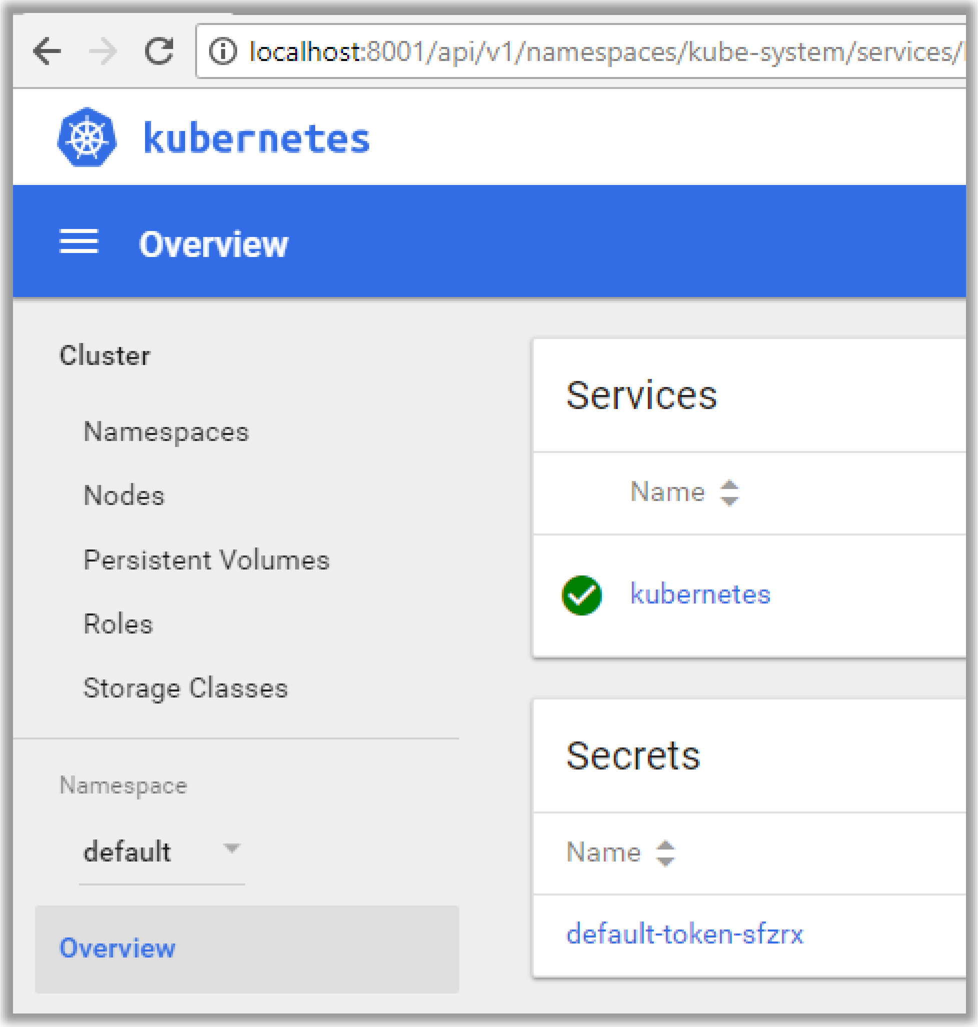
Set up the KubeCtl tool

10. Make a folder called “kubectl” under your “C:\” then run:
    $ az acs kubernetes install-cli --install-location=C:\kubectl\kubectl.exe
11. Put kubectl on your PATH:



```
C:\Program Files (x86)\Microsoft SDKs\Azure\CLI2\wbin
C:\Program Files\Docker\Docker\Resources\bin
%SystemRoot%\system32
%SystemRoot%
%SystemRoot%\System32\Wbem
%SYSTEMROOT%\System32\WindowsPowerShell\v1.0\
C:\Program Files (x86)\Skype\Phone\
C:\Program Files\dotnet\
C:\Program Files\Microsoft SQL Server\130\Tools\Binn\
C:\Program Files\Git\cmd
C:\Program Files\nodejs\
C:\Program Files\Microsoft\Web Platform Installer\
C:\Program Files (x86)\Microsoft SQL Server\Client SDK\ODBC\130\To...
C:\Program Files (x86)\Microsoft SQL Server\140\Tools\Binn\
C:\Program Files (x86)\Microsoft SQL Server\140\DTS\Binn\
C:\Program Files (x86)\Microsoft SQL Server\140\Tools\Binn\Manage...
C:\ProgramData\chocolatey\bin
C:\Program Files\Minikube
C:\kubectl
```

12. Get the key that will enable you to interact directly with AKS from the command line:
    $ az acs kubernetes get-credentials --resource-group=ABC-K8s --name=ABC-Kube-Cluster
    You should see the output as follows:
    Merged "k8s-kubemgmt" as current context in C:\Users\bernardo\.kube\config
13. Start the kubectl proxy:
    $ kubectl proxy
    You should see output as follows: “Starting to serve on 127.0.0.1:8001”
    Browse to http://localhost:8001/ui



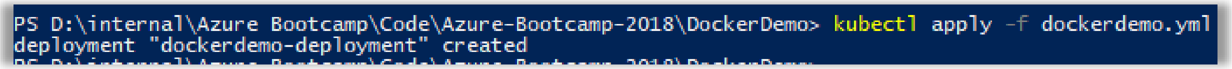
localhost:8001/api/v1/namespaces/kube-system/services/

Deploy our microservice to AKS

14. Add a deployment file that we will use to upload to Kubernetes – I called mine dockerdemo.yml:

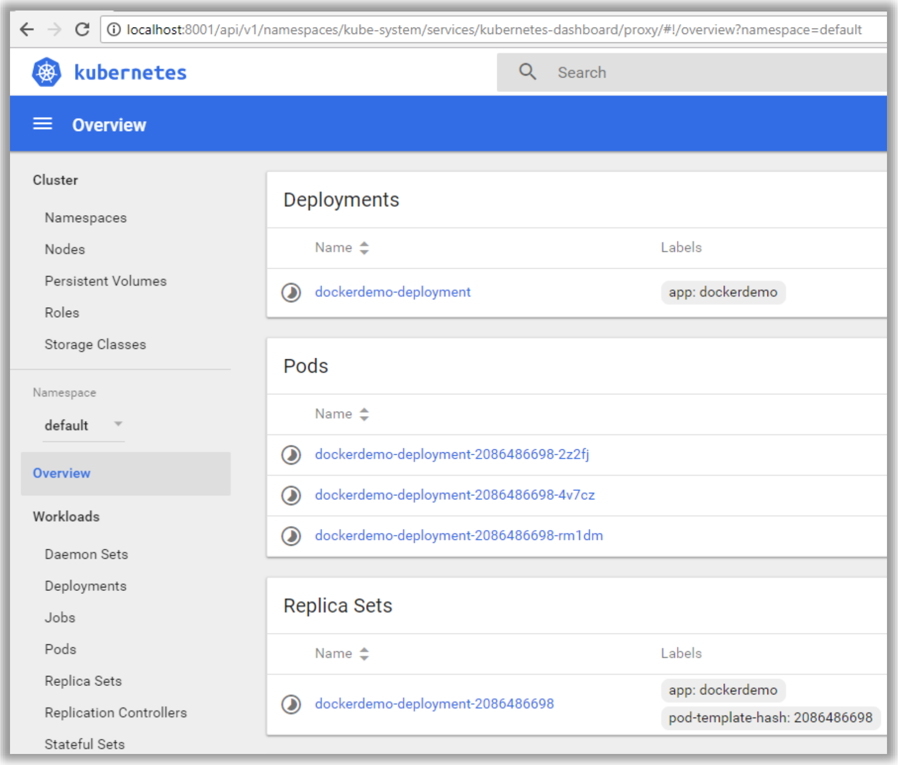
```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: abcdockerdemo-deployment
spec:
  replicas: 3
  template:
    metadata:
      labels:
        app: abcdockerdemo
    spec:
      containers:
      - name: dockerdemoapi
        image: <your docker hub namespace>/dockerdemoapi:latest
        ports:
        - containerPort: 80
      - name: inventorydb
        image: <your docker hub namespace>/inventorydb:latest
        ports:
        - containerPort: 1433
```

15. Deploy to AKS:
    $ kubectrl apply -f dockerdemo.yml

    ```
    PS D:\internal\Azure Bootcamp\Code\Azure-Bootcamp-2018\DockerDemo> kubectl apply -f dockerdemo.yml
    deployment "dockerdemo-deployment" created
    ```

    You should be able to see the Kubernetes pods, etc, deploying via the UI – the images are being pulled from Docker Hub:

    Deployments

Name	Labels
dockerdemo-deployment	app: dockerdemo

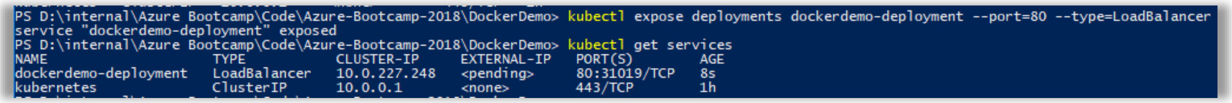
    Pods

Name
dockerdemo-deployment-2086486698-2z2fj
dockerdemo-deployment-2086486698-4v7cz
dockerdemo-deployment-2086486698-rm1dm

    Replica Sets

Name	Labels
dockerdemo-deployment-2086486698	app: dockerdemo pod-template-hash: 2086486698

16. Request a load balancer setup so that we can expose the API on the internet:
    $ kubectl

    ```
    PS D:\internal\Azure Bootcamp\Code\Azure-Bootcamp-2018\DockerDemo> kubectl expose deployments dockerdemo-deployment --port=80 --type=LoadBalancer
    service "dockerdemo-deployment" exposed
    PS D:\internal\Azure Bootcamp\Code\Azure-Bootcamp-2018\DockerDemo> kubectl get services
    NAME                    TYPE           CLUSTER-IP     EXTERNAL-IP   PORT(S)        AGE
    dockerdemo-deployment   LoadBalancer   10.0.227.248   <pending>     80:31019/TCP   8s
    kubernetes              ClusterIP      10.0.0.1       <none>        443/TCP        1h
    ```

    Get the list of running services, as shown above – you should see that there is an IP address being applied to the "dockerdemo-deployment" (as above). This takes a little while to apply.
    $ kubectl get services
17. All going well you should be able to see a result in our "/api/values" and "/api/database" endpoints.

Workshop 3 – Make your own Container Registry (ACR)

We're going to make our own container registry on ACR, push our images to it and connect our AKS instance to it…