PitStop Workshop

Welcome to the PitStop workshop! In this workshop you will learn how to build Microservices based systems using .NET Core and Docker. In most workshops you start from scratch, but in this workshop you will actually start with a complete working solution. You will learn by adding functionality to the solution.

Lab 0: Preparation

There are some prerequisites for this workshop. First you need an active Internet connection. Additionally you will need to install the following software on your laptop:

- Docker Community Edition (CE)
- Visual Studio 2017 (Community or Code)
- .NET Core SDK
- (optional) Git client

> If you already have satisfied these prerequisites, you can skip Lab 0 and go directly to Lab 1.

Step 0.1: Install prerequisites

Install the following software (if not already installed) on your laptop:

Docker CE

Download link: Docker Community Edition (CE).

On Windows, you need Hyper-V to be enabled on your machine in order to install Docker for Windows CE. If you have not enabled Hyper-V, do so now. Here you will find a description of how to enable Hyper-V on Windows. Make sure to double-check the prerequisites.

For downloading Docker CE, you need to login with your Docker Id. Create one if you don't already have a Docker Id.

During the installation of Docker CE, do not switch to Windows containers. We will only use Linux containers. After the installation you need to log out and login again (sometimes reboot your machine).

After the installation, start the Docker engine by double clicking the Docker for Windows icon.

Visual Studio

This workshop assumes you are working with Visual Studio Code (which runs on Windows, Linux and OSX). If you use Visual Studio Community, there will be some minor differences.

Download link: Visual Studio 2017 (Community or Code)

.NET Core SDK

Install the .NET Core SDK 2.1.

Download link: .NET Core SDK

Step 0.2: Sources

Enclosed in the .zip file where you found this preperation you also received the sources of the application (named PitStop) we are going to use. You can find these sources in the /src folder of the .zip file. Unpack this .zip so you have the following structure:

- root
  - src
  - workshop

Lab 1: Run the applicaton

In this lab we'll make sure you can run PitStop on your machine. This will involve the following activities:

This would be a good time to walk through the solution and see what's in there.

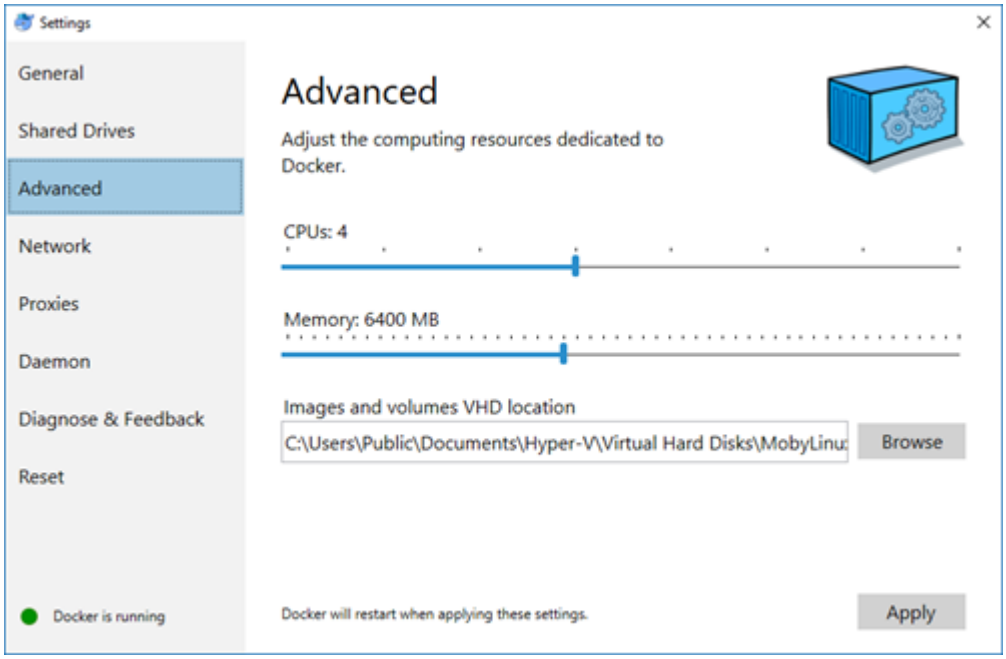
Step 1.1: Build the Docker images

In order to run the application you need to take several steps. This description assumes you're developing on a Windows machine using Visual Studio 2017 or Visual Studio Code and have extracted the .\src folder from the .zip file you received.

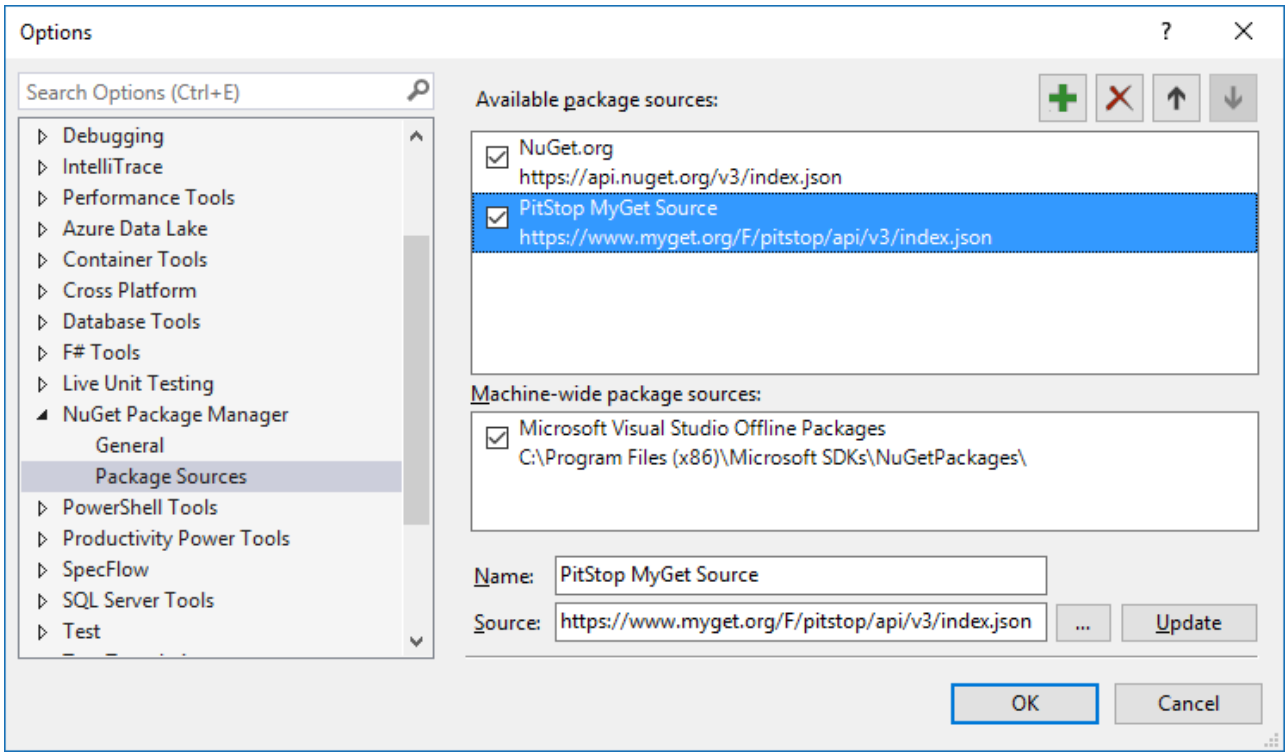
Make sure your have satisfied the Prerequisites.

> In the `docker-compose.yml` file in the root of the solution folder there are some credentials specified for components that need them. These are also used by the different services that use these components (specified in config files): SQL Server login: sa / 8jkGh47hnDw89Haq8LN2, Rabbit MQ login: rabbitmquser / DEBmbwkSrzy9D1T9cJfa

- Satisfy prerequisites
    - Make sure you have Docker installed and running smoothly on your machine. This sample only uses Linux based containers. Also make sure everything is configured correctly in order to pull Docker images from the public Docker hub.
    - Increase the amount of memory dedicated to Docker to at least 4 GB. You can do this on the *Advanced* tab of the Docker settings dialog:



- Open the PitStop solution in Visual Studio or in Visual Studio Code.
- When using Visual Studio, add a NuGet source To prevent project-references between projects in the solution, I've used a public MyGet feed for shared components. The URI for this feed is: `https://www.myget.org/F/pitstop/api/v3/index.json`. This feed contains the *Infrastructure.Messaging* package. Open Visual Studio and configure the NuGet sources and add the MyGet feed:



.

> The MyGet feed is read-only.

- Rebuild solution To make sure everything is setup correctly, do a Rebuild All of the solution. This will also restore all the NuGet packages used throughout the solution. If no errors occur, you're good to go.
- Build docker images Open up a Powershell window and go to the `Pitstop/src` folder. Then execute the `RebuildAllDockerImages` script. This will rebuild all the Docker images for all the projects. Watch the output for any errors. After the images are built, you could check whether they are all there using the `docker images` command. This should yield something like this:

```
PS C:\dev\pitstop\src\CustomerEventHandler> docker images
REPOSITORY                                  TAG                 IMAGE ID            CREATED             SIZE
pitstop/customereventhandler                latest              26db5e5c1e55        2 minutes ago       182MB
pitstop/webapp                              latest              3990c595eab9        33 hours ago        265MB
pitstop/workshopmanagementeventhandler      latest              08d956f9b400        33 hours ago        190MB
pitstop/timeservice                         latest              ba538b7306e9        33 hours ago        182MB
pitstop/notificationservice                 latest              97e341117f3e        33 hours ago        187MB
pitstop/invoiceservice                      latest              b72f1512b2ca        33 hours ago        187MB
pitstop/auditlogservice                     latest              2f7959fba0e9        33 hours ago        182MB
pitstop/apigateway                          latest              b26547749f6f        33 hours ago        272MB
pitstop/workshopmanagementapi               latest              61ea6a13a78c        33 hours ago        269MB
pitstop/customermanagementapi               latest              7df7e5acb5b7        33 hours ago        269MB
pitstop/vehiclemanagementapi                latest              57352622f3b6        33 hours ago        269MB
datalust/seq                                latest              620f5f268f7e        6 days ago          668MB
rabbitmq                                    3-management        75472a5c510b        9 days ago          149MB
microsoft/dotnet                            2.1-aspnetcore-runtime 40d759655ea3     11 days ago         255MB
microsoft/dotnet                            2.1-runtime         cc240a7fd027        11 days ago         180MB
microsoft/dotnet                            2.1-sdk             e1a56dca783e        11 days ago         1.73GB
microsoft/dotnet                            latest              e1a56dca783e        11 days ago         1.73GB
microsoft/mssql-server-linux                latest              885d07287041        13 days ago         1.45GB
djfarrelly/maildev                          latest              dfbc3576b8d5        9 months ago        68.8MB
PS C:\dev\pitstop\src\CustomerEventHandler>
```

As part of the `RebuildAllDockerImages` script, two Docker volumes are created. One for the SQL Server data and one for the RabbitMQ data. This ensures that data for these infrastructural components survives restarts of the Containers. If you don't use the `RebuildAllDockerImages` script, you can create the volumes by hand:

- `docker volume create sqlserverdata`
- `docker volume create rabbitmqdata`

Step 1.2: Run the application

This will describe how to run the Pitstop solution using Docker-Compose.

Execute the following steps to start the application:

- Make sure your have satisfied the Prerequisites and you have built all the Docker images as described in Building the Docker images. You can also pull the images from the Pitstop repo available in Docker Hub.
- Open up a Powershell window and go to the `Pitstop/src` folder.
- Issue the following command: `docker-compose up`. This will start the solution with a single instance of the API services.

Optional steps

The steps described below are optional steps to test different aspects of the application.

Scaling the services

If you want to start the solution with multiple instances of the API services running, use the following command: `docker-compose up -d --scale customermanagementapi=2 --scale vehiclemanagementapi=2 --scale workshopmanagementapi=3`. This uses the *scale* argument to specify the number of instances of each services that must be started. The docker-compose networking stack will automatically load-balance the requests over the available instances. From the client perspective nothing needs to change. It can still communicate with a service by using the container-name specified in the `docker-compose.yml` file (e.g. `workshopmanagementapi`) as if there was only 1 instance. But when you look at the started containers, you will see multiple instances running with a name that is postfixed with a number (e.g. `workshopmanagementapi_1`, `workshopmanagementapi_2`).

Running with APIs exposed on localhost

In order to make sure the APIs are reachable from localhost, start the solution using the *docker-compose.local.yml* override file. You do this by starting the solution using the following command: `docker-compose -f docker-compose.yml -f docker-compose.local.yml up`. This will make sure that the ports used by the API services are exposed to the host so you can access them directly from the browser.

You can also use the *StartSolutionWithLocalAPIs.ps1* / *StartSolutionWithLocalAPIs.sh* scripts to start the solution.

Now you can access the individual APIs in the system. You can use the test UIs that are auto-generated by Swashbuckle.

The following URLs can be used:

API	URL
CustomerManagement	http://localhost:5100/swagger
VehicleManagement	http://localhost:5000/swagger
WorkshopManagement	http://localhost:5200/swagger

> This way of running the application is only possible if you run 1 instance of each service. If you run more than one instance, this will result in an error because multiple instances would be exposed on the same port on localhost.

Step 1.3: Get to know the solution

To test the application you need to open the following web-pages:

- The PitStop web-application: http://localhost:7000.

Now you can follow the following scenario (make sure you fill all the fields in the entry-forms):

- Register a new customer on the *Customer Management* screen.
- Register a new Vehicle for this customer on the *Vehicle Management* screen.
- Register a couple of Maintenance Jobs for the vehicle on the *Workshop Management* screen.