4.5 Scale Your App to Run on Multiple Containers



This section will guide you to:

* Setup Docker Compose and Docker Desktop in Windows

**Development Environment**

* Windows
* Docker

This guide has five subsections, namely:

4.5.1 Setting up Docker Desktop in Windows

4.5.2 Setting up Docker Compose

4.5.3 Downloading the sample eShopOnContainers .NET application

4.5.4 Setting up a docker-compose.yml file to deploy it on multiple containers

4.5.5 Pushing the code to GitHub repositories

**Step 4.5.1:** Setting up Docker Desktop in Windows

* Docker is already installed in your practice lab. Refer to **DotNet Lab guide: Phase 4** for more information.
* To test the installation, open the Windows Command Prompt and type *docker –version*
* This will show the installed version of Docker.

**Step 4.5.2:** Setting up Docker Compose

* Docker Compose comes installed with Docker.
* To check if it is installed, at the command prompt type in the following command:

*docker-compose --version*

* If it shows a version, then it is already installed, and the rest of the installation steps may be skipped.
* Start an elevated PowerShell terminal.
* Type *[Net.ServicePointManager]::SecurityProtocol = [Net.SecurityProtocolType]::Tls12*
* Download the current stable release of Docker Compose: *Invoke-WebRequest "https://github.com/docker/compose/releases/download/1.25.3/docker-compose-Windows-x86\_64.exe" -UseBasicParsing -OutFile $Env:ProgramFiles\Docker\docker-compose.exe*
* To test the successful installation type *docker-compose –version*

**Step 4.5.3:** Downloading the sample eShopOnContainers .NET application

* Open Powershell and download the sample application from GitHub using the following command:

*git clone* [*https://github.com/dotnet-architecture/eShopOnContainers.git*](https://github.com/dotnet-architecture/eShopOnContainers.git)

* Load it up in Visual Studio and run it locally to make sure it is working

**Step 4.5.4:** Setting up a docker-compose.yml file to deploy it on multiple containers

* Create a docker-custom.yml file in the src subfolder of the application with the following script:

*version: '3.4'*

*services:*

*webmvc:*

*image: eshop/webmvc*

*environment:*

*- CatalogUrl=http://catalog.api*

*- OrderingUrl=http://ordering.api*

*- BasketUrl=http://basket.api*

*ports:*

*- "5100:80"*

*depends\_on:*

*- catalog.api*

*- ordering.api*

*- basket.api*

*catalog.api:*

*image: eshop/catalog.api*

*environment:*

*- ConnectionString=Server=sql.data;Initial Catalog=CatalogData;User Id=sa;Password=your@password*

*expose:*

*- "80"*

*ports:*

*- "5101:80"*

*depends\_on:*

*- sql.data*

*ordering.api:*

*image: eshop/ordering.api*

*environment:*

*- ConnectionString=Server=sql.data;Database=Services.OrderingDb;User Id=sa;Password=your@password*

*ports:*

*- "5102:80"*

*depends\_on:*

*- sql.data*

*basket.api:*

*image: eshop/basket.api*

*environment:*

*- ConnectionString=sql.data*

*ports:*

*- "5103:80"*

*depends\_on:*

*- sql.data*

*sql.data:*

*environment:*

*- SA\_PASSWORD=your@password*

*- ACCEPT\_EULA=Y*

*ports:*

*- "5434:1433"*

*basket.data:*

*image: redis*

* In the Powershell, navigate to the folder containing docker-custom.yml and run the following command to run the yml script type:

*docker-custom.yml up --build*

* The deployed application will have each of its api services running from a different container. The database is also being served from a Redis cache container

**Step 4.5.5:** Pushing the code to your GitHub repositories

Open your command prompt and navigate to the folder where you have created your files.

cd <folder path>

Initialize your repository using the following command:

git init

Add all the files to your git repository using the following command:

git add .

Commit the changes using the following command:

git commit -m “Changes have been committed.”

Push the files to the folder you created initially using the following command:

git push -u origin master