Demo Script: Docker Booth Demos

Steve Lasker

[Steve.Lasker@Microsoft.com](mailto:Steve.Lasker@Microsoft.com)

<http://blogs.msdn.com/SteveLasker>

# Docker **build**

# Demo Reset

From GitHub, revert changes for **works everywhere**

# Production Environment

|  |  |
| --- | --- |
| **Demo step** | **Talk track & notes** |
| ***Look at Production -*** Not Working …? | |
| Open <http://azuredemo/> | Let’s take a look at this site deployed into production  It’s a standard ASP.NET project |
| Navigate to About  Point to **HostName** and **OS** | Notice we’re missing some elements from being displayed |

## Works Fine on My Machine

|  |  |
| --- | --- |
| **Demo step** | **Talk track & notes** |
| ***View in Development Mode*** | |
| Open the project | Here’s our project that we have deployed |
| F5 | We’ll launch the project under IIS Express |
| Click the About box | Notice we’re running on Windows, we can see our machine name  ***Works On My Machine*** |
|  | We have this project automatically deployed, using docker containers and Linux.  We obviously have a problem, but what could it be? How can we debug the production environment locally? |
| Add Docker Support | We’ll add docker support so that we can run our project in a Linux container and validate all is working as expected |
| Point to the Docker Target | Notice we now have Docker set as our default target |
| Hit F5 | We’re now running our code in a container  Because we’re running under the debug configuration, we’re using something called volume mapping  Our app is running in a container – another instance of a “server”. However, the code from that container is actually using volume mapping to our host os – my laptop.  Visual Studio is still editing the files, but the container thinks they are local to the container |
| Browse to the About Page | Hmmm, notice the HostName, OS Processor Architecture are all blank  ***Doesn’t work on my machine*** |

# Edit & Refresh – Works on My Machine

|  |  |
| --- | --- |
| Open Views\Home\About.cshtml | Let’s make some changes and see how we still get our normal Edit & Refresh scenarios |
| Add to the bottom of the about page <p>running in a container</p> | We’ll first simply add some text to the bottom of the page |
| Hit Refresh in the browser | Notice we quickly see the changes we just made |
| Open Controllers\HomeController.cs  Replace the About method with the 2nd snippet | Let’s make a code change, replacing our code with more platform resilient code, which we’ll test of course |
| Hit refresh again | Notice we’re now seeing the OS and Machine information  **Works on My Machine, and the Production Machine** |

# Deploy to Azure, using VSTS

|  |  |
| --- | --- |
| **Demo step** | **Talk track & notes** |
|  | |
| **Launch** [https://devinacontainer.visualstudio.com](https://devinacontainer.visualstudio.com/DefaultCollection/WorksOnMyMachine/)  Click build  Select Web Build  Select Edit to show the build steps | Let’s take a look at the automated CI /CD configuration we have setup  Looking in our configuration we see we have several build steps   * We build the image from the release dockerfile * Compose the collection of services to run our tests   + Currently using run as we work out some kinks with compose * We would run some functional tests * If successful   + Push the image with the build #   + Tag the image latest   + Push the newly tagged image – just a reference   + Force a pull of latest into our staging   + Compose up into our Staging environment |
| **Using GitHub – Check in the change** | We’ll use GitHub to check-in the change |
| **Navigate to the Web Build – Queued** | We can see our automated build kick in  We can even watch the logs as it processes |
| **Navigate to** [**http://**azuredemo](http://azuredemo) | Now we see the demo running in our deployed environment  This could be your staging server, where your team manually validates it, or it can be deployed straight to ACS as well |

# Demo environment setup & prereqs

## Install and Configure Docker For Windows Beta

[Setting Up Docker For Windows (D4W)](onenote:Piñata.one#Setting%20Up%20Docker%20For%20Windows%20(D4W)&section-id={3A79CD1D-D9A5-484C-85FE-7B734C48A4F4}&page-id={43D2985A-D59C-455D-A413-69A585BBC0C8}&end&base-path=https://microsoft.sharepoint.com/teams/CPT/AzureTools/Shared%20Documents/Docker/Docker%20Investiga)

# Add Hosts Entries

Add the following hosts entries to make it easier to connect to machines in the browser:

10.0.75.2 docker

13.93.213.101 azurebuild

13.91.255.86 azuredemo

13.93.215.124 azurehost

# Creating Docker Hosts in Azure

Using Docker-machine 0.7.0

docker-machine create -d azure --azure-subscription-id [azuresubid] --azure-resource-group build2016demos --azure-open-port 80 --azure-open-port 5000 --azure-open-port 8000-8200 azuredemo

# Open the project

<https://github.com/SteveLasker/WorksOnMyMachine>

# Cached Docker Images for ASP.NET

Run the following in a PowerShell prompt

Write-Host Pulling ASP.NET RC1 DNX Debug Image w/Nugets -foregroundcolor "Green"

Write-Host Executing: docker pull stevelasker/aspnet:1.0.0-rc1-update1-dnx-debug-wnuget -foregroundcolor "yellow"

docker pull stevelasker/aspnet:1.0.0-rc1-update1-dnx-debug-wnuget

Write-Host Renaming to: microsoft/aspnet:1.0.0-rc1-update1-debug -foregroundcolor "Green"

Write-Host Executing: docker tag stevelasker/aspnet:1.0.0-rc1-update1-dnx-debug-wnuget microsoft/aspnet:1.0.0-rc1-update1-debug -foregroundcolor "yellow"

docker tag stevelasker/aspnet:1.0.0-rc1-update1-dnx-debug-wnuget microsoft/aspnet:1.0.0-rc1-update1-debug

Write-Host Pulling ASP.NET RC1 DNX Release Image w/Nugets -foregroundcolor "Green"

Write-Host Executing: docker pull stevelasker/aspnet:1.0.0-rc1-update1-dnx-release-wnuget -foregroundcolor "yellow"

docker pull stevelasker/aspnet:1.0.0-rc1-update1-dnx-release-wnuget

Write-Host Renaming to: microsoft/aspnet:1.0.0-rc1-update1-release -foregroundcolor "Green"

Write-Host Executing: docker tag stevelasker/aspnet:1.0.0-rc1-update1-dnx-release-wnuget microsoft/aspnet:1.0.0-rc1-update1-release -foregroundcolor "yellow"

docker tag stevelasker/aspnet:1.0.0-rc1-update1-dnx-release-wnuget microsoft/aspnet:1.0.0-rc1-update1-release

Write-Host Cleaning up faked image names -foregroundcolor "Green"

Write-Host Executing: docker rmi stevelasker/aspnet:1.0.0-rc1-update1-dnx-release-wnuget -foregroundcolor "yellow"

docker rmi stevelasker/aspnet:1.0.0-rc1-update1-dnx-release-wnuget

Write-Host Executing: docker rmi stevelasker/aspnet:1.0.0-rc1-update1-dnx-debug-wnuget -foregroundcolor "yellow"

docker rmi stevelasker/aspnet:1.0.0-rc1-update1-dnx-debug-wnuget

Write-Host Current Images -foregroundcolor "Green"

docker images

# Demo Reset

**docker rm -f $(docker ps -a -q)**

**docker rmi busybox**

## Cache Images

**docker pull microsoft/aspnet**

**docker pull Ubuntu**

**docker pull tutum/haproxy**

# Reset All Images

Only necessary to clear all, and start from scratch.

**docker rm -f $(docker ps -a -q)**

**docker rmo -f $(docker images -q)**