.NET Core 2.0 on Red Hat® Enterprise Linux Cheat Sheet

RED HAT® DEVELOPER PROGRAM

Table of contents

1. Introduction 1
2. Assumptions 1
3. Installation 1
4. "Pay to Play" model of .NET Core 2
5. Portable vs. standalone applications . . 2
6. Hello World 3
7. .NET CLI commands 3
8. .NET and Linux containers 5
9. Creating an ASP.NET MVC website . . . 5
10. Valuable resources 7

Introduction

The .NET Framework has been around since 2002, supporting Windows developers in many programming languages. .NET Core, made available in 2016, is the latest .NET platform, expanded to run on Windows, macOS and Linux. A Windows developer can now, for example, write code that runs on Red Hat Enterprise Linux (RHEL) with almost zero learning curve.

While support for desktop applications is not included in .NET Core, web sites, web services, and even Linux daemons can be written in the supported .NET languages: C#, Visual Basic, and F#.

Because .NET can now run on Linux, the world of microservices has opened up to .NET developers as well. Running a .NET program in a Linux container is as simple as it is in any other language.

This cheat sheet will cover the basics in installing .NET on Red Hat Enterprise Linux (RHEL), how to get a simple program running, and how to run a program in a Linux container. Finally, a list of valuable resources is included.

Assumptions

This cheat sheet assumes you are using the Red Hat Development Suite https://developers.redhat.com/products/devsuite/download/ to install and run Red Hat Enterprise Linux, either on a RHEL machine or on your Virtual Machine. If you are using a different machine, any password used in the installation, following, may differ.

Installation

To install the .NET Core 2.0 SDK on RHEL, use the following eight commands:

```
sudo subscription-manager register
sudo subscription-manager list --available
```

(Note the Pool ID near the end of the list; you'll need this for the next step)

```
sudo subscription-manager attach --pool={Pool ID from previous step}
```

One of the following for Red Hat Enterprise Linux 7 Server, Red Hat Enterprise Linux 7 Workstation, or HPC Compute Node:

```
sudo subscription-manager repos --enable=rhel-7-server-dotnet-rpms
sudo subscription-manager repos --enable=rhel-7-workstation-dotnet-rpms
sudo subscription-manager repos --enable=rhel-7-hpc-node-dotnet-rpms
sudo yum install -y scl-utils
sudo yum install -y rh-dotnetcore11
echo 'source scl_source enable rh-dotnet20' >>~/.bashrc
```

You will need to exit your terminal session and start a new one to have access to .NET Core.

The "Pay to Play" model of .NET Core

.NET Core is small. For example, the .NET Core SDK on a Windows 10 machine takes less than 200 MB. This is because the new development model uses only the bits you need to build and run your code. This strategy, sometimes called "Pay to Play", means you must specifically call out pieces of .NET Core in your code, and code your dependencies (in the project file, e.g. myapp.csprog) accordingly. Likewise, if you remove a reference to a library in your code, you can benefit from a smaller code footprint by also removing the dependency from the project file.

Portable vs. Standalone Applications

When you build a .NET application, you have two output options: Portable (the default) or Standalone.

Portable Application

By default, running dotnet build will build your program as a Portable Application. This means the compiler will create {appname}.dll; for example, helloworld.dll. This DLL can then be copied to any machine hosting an installation of .NET Core and it will run, regardless of operating system. For example, you could copy the DLL from a Mac onto a Linux machine (which has .NET Core installed) and it will execute.

Standalone Application

.NET Core allows you to compile an application so that it will run on a machine that does not have .NET installed. Further, you can cross-compile from any operating system to any operating system. For example, you can compile your application on your Windows 10 machine and then install and run it on a Linux machine that does not have .NET Core installed. This powerful feature of .NET Core does not exist in the previous .NET Framework. To create a standalone application, you use the dotnet publish command.

Note that the standalone application does not create an ".exe" file (Yet. This functionality, referred to as ".NET Native", is scheduled for later in 2018). Like the Portable application, it creates a DLL that is run by the dotnet host. The standalone application avoids the need for a .NET Core installation by copying all of the necessary bits to the directory where you build the application. That is, you are not copying one file to the target; you must copy the entire directory and any sub-directories.

The "Hello World" program

Get a simple "Hello World" program running in four commands:

```
mkdir helloworld && cd helloworld
dotnet new
dotnet run
```

dotnet CLI commands

dotnet --help

The '--help' flag will display help for the lowest-level available based on the 'dotnet' command you've typed. 'dotnet --help' gives you broad help about the 'dotnet' command, while 'dotnet new --help' gives you more detailed help about the 'dotnet new' command, and so on.

dotnet new [options]

The 'dotnet new' command is used to create a new project. This function uses the dotnet templating engine, which is continually evolving. It can be found at https://github.com/dotnet/templating.

For example, to create an MVC ASP.NET web site, you would run the following command:

```
dotnet new mvc
```

This creates a web site using C# as the language. Adding the --language F# flag will create source code in F#.

dotnet restore

The 'dotnet restore' command will retrieve the dependency libraries needed for your application as defined in the project file. By default, the command will download the libraries from https://www.nuget.org. Note that 'dotnet restore' will run automatically when creating a new project unless you use the '--no-restore' flag.

dotnet add

The 'dotnet add' command is used to insert a reference into your project file. While you can, if you wish, manually edit your project file, this command makes it easier to add a reference. For example, if you wish to reference the NuGet package "Json.NET" in your project, you would use the following command:

```
dotnet add package Newtonsoft.Json
```

.NET and Linux Containers

You can run your .NET code in a Linux container just as you can any code that runs in Linux. This allows you to create ready-to-run images that are isolated from the host operating system, can easily be transported, and start in just a few seconds.

Simple web service

Consider the following, very small web service. Using the default code that is generated using the 'dotnet new webapi' command, the following sections will describe how to create and run a Linux image.

Creating the source code

To create the source code, create a directory, move into it, and run the proper command:

```
mkdir myexample && cd myexample
dotnet new webapi
```

Creating the Dockerfile

A configuration file, 'Dockerfile', is required to build the Linux image for this program. While you can use the 'microsoft/dotnet' image to run your program, it won't be using RHEL. To use RHEL to run your application, your Dockerfile should follow the following example:

```
FROM registry.access.redhat.com/dotnet-20-runtime-rhel7
ADD bin/Release/netcoreapp2.0/rhel.7-x64/publish/. /app/
WORKDIR /app/
EXPOSE 5000
CMD ["scl", "enable", "rh-dotnet20", "--", "dotnet", "myexample.dll"]
```

Creating the runtime script

In order to use the RHEL runtime in a Linux container, you need to use a parameterized script. Add this script in a file called "standalone.sh" to your project's root directory:

```
#!/bin/bash
ASSEMBLY=webapi.dll
SCL=rh-dotnet20
DIR="$(dirname "$(readlink -f "$0")")"
scl enable $SCL -- dotnet "$DIR/$ASSEMBLY" "$@"
```

To use this script, you'll need to add the following section to your project file:

```
<ItemGroup>
<None Update="standalone.sh" Condition="'$(RuntimeIdentifier)'
== 'rhel.7-x64' and '$(SelfContained)' == 'false'"
CopyToPublishDirectory="PreserveNewest" />
</ItemGroup>
```

Creating the executable

In order to use this Dockerfile, you must first publish your .NET application into the correct directory. The above Dockerfile expects the following publish command:

```
dotnet publish -f netcoreapp2.0 -c Release -r rhel.7-x64 --self-contained
false /p:PublishWithAspNetCoreTargetManifest=false
```

Building the image

The following command will build the image. Note that the image name does not need to match the application name.

```
docker build -t webapi .
```

Running the container

Use the following command to run the docker container:

```
docker run -d -p 8080:8080 webapi
```

Future image builds

To rebuild the image, use the following five steps from the root directory of your project:

```
rm -rf obj/
rm -rf bin/
dotnet restore
dotnet publish -f netcoreapp2.0 -c Release -r rhel.7-x64 --self-contained
 false /p:PublishWithAspNetCoreTargetManifest=false
docker build -t webapi .
```

Creating an ASP.NET MVC web site

Creating a simple ASP.NET MVC web site is done by using the dotnet new command, as follows:

```
dotnet new mvc
```

This creates the source files and folders necessary. Worth noting are the MVC routing entries near the bottom of Startup.cs:

```
app.UseMvc(routes =>
{
  routes.MapRoute(
    name: "default",
    template: "{controller=Home}/{action=Index}/{id?}");
});
```

Also notice that in order to use static (i.e *.html) files, you must specify that intention with the dependency injection code, also near the bottom of Startup.cs:

```
app.UseStaticFiles();
```

Valuable Resources

The following web sites are excellent sources of information regarding .NET Core development:

dot.net	This vanity URL will redirect you to Microsoft's official .NET Core page. From there you can download and install .NET Core, find documentation, etc.
live.asp.net	This URL will direct you to the weekly ASP.NET Core Community Standup, a live webcast (also recorded and available on YouTube) of the newest happenings regarding ASP.NET and .NET Core.
redhatloves.net	This vanity URL directs you to the .NET on Linux section of the Red Hat Developer Program. This program contains resources such as books, blog posts, videos, and free software -- including the Red Hat Development Suite. This suite includes: RHEL, Virtual Box, Vagrant, docker, kubernetes, and OpenShift.
github.com/dotnet	The github home of the .NET Foundation.
github.com/aspnet	The home of the ASP.NET Core code, including EntityFramework (which is included in this repo but is not a part of ASP.NET).

To install a local version of Red Hat Enterprise Linux or OpenShift Container Platform on your desktop/laptop, go to https://developers.redhat.com/products/devsuite/hello-world/

DON SCHENCK is developer who has seen it all. He is a Microsoft MVP and currently Director of Developer Experience at Red Hat, with a focus on Microsoft .NET on Linux. His mission is to connect .NET developers with the Linux and open source communities. Prior to Red Hat, Don was a Developer Advocate at Rackspace where he was immersed in cloud technology. He still enjoys cooking and studying human behavior, and still hates the designated hitter rule.

Don's overarching belief is this: "A program is not a communication between a developer and a machine; it's a communication between a developer and the next developer."

@DonSchenck github.com/DonSchenck linkedin.com/in/DonSchenck