.NET Core 2.0

Getting Started Guide
.NET Core 2.0 Getting Started Guide
Abstract

.NET Core is a general purpose development platform featuring automatic memory management and modern programming languages. It allows users to build high-quality applications efficiently. .NET Core is available in Red Hat Enterprise Linux (RHEL 7) and OpenShift Container Platform via certified containers. .NET Core offers: the ability to follow a microservices-based approach, where some components are built with .NET and others with Java, but all can run on a common, supported platform in Red Hat Enterprise Linux and OpenShift Container Platform. The capacity to more easily develop new .NET Core workloads on Microsoft Windows. Customers can deploy and
run on either Red Hat Enterprise Linux or Windows Server. In a heterogeneous data center, where the underlying infrastructure is capable of running .NET applications without having to rely solely on Windows Server. .NET Core 2.0 is supported on Red Hat Enterprise Linux 7 and OpenShift Container Platform versions 3.3 and later.
# Table of Contents

CHAPTER 1. .NET CORE 2.0 ON RED HAT ENTERPRISE LINUX .................................................. 3  
 1.1. INSTALL AND REGISTER RED HAT ENTERPRISE LINUX ........................................... 3  
 1.2. INSTALL .NET CORE ................................................................................................. 4  
 1.3. CREATE AN APPLICATION ....................................................................................... 5  
 1.4. PUBLISH APPLICATIONS .......................................................................................... 5  
  1.4.1. Publish .NET Core Applications ........................................................................... 5  
  1.4.2. Publish ASP.NET Core Applications ..................................................................... 6  
  1.5. RUN APPLICATIONS ON DOCKER ......................................................................... 7  

CHAPTER 2. .NET CORE 2.0 ON RED HAT OPENSHIFT CONTAINER PLATFORM .................... 9  
 2.1. INSTALL IMAGE STREAMS .......................................................................................... 9  
 2.2. DEPLOY APPLICATIONS ............................................................................................ 10  
 2.3. ENVIRONMENT VARIABLES ...................................................................................... 11  
 2.4. SAMPLE APPLICATIONS ............................................................................................ 12  
 2.5. CREATE A RUNTIME IMAGE ....................................................................................... 13  

APPENDIX A. REVISION HISTORY ....................................................................................... 15
CHAPTER 1. .NET CORE 2.0 ON RED HAT ENTERPRISE LINUX

This Getting Started Guide describes how to install .NET Core 2.0 on Red Hat Enterprise Linux (RHEL). See Red Hat Enterprise Linux documentation for more information about Red Hat Enterprise Linux 7.

1.1. INSTALL AND REGISTER RED HAT ENTERPRISE LINUX

1. Install RHEL 7 using one of the following images:
   - Red Hat Enterprise Linux 7 Server
   - Red Hat Enterprise Linux 7 Workstation
   - Red Hat Enterprise Linux for Scientific Computing

2. Register the system by following the appropriate steps in Registering and Unregistering a System in the Red Hat Subscription Management document. You can also use the following command to register the system.
   
   ```
   $ sudo subscription-manager register
   ```

3. Display a list of all subscriptions that are available for your system and identify the pool ID for the subscription.
   
   ```
   $ sudo subscription-manager list --available
   ```
   This command displays the subscription name, unique identifier, expiration date, and other details related to it. The pool ID is listed on a line beginning with Pool ID.

4. Attach the subscription that provides access to the dotNET on RHEL repository. Use the pool ID you identified in the previous step.
   
   ```
   $ sudo subscription-manager attach --pool=<appropriate pool ID from the subscription>
   ```

5. Enable the .NET Core channel for Red Hat Enterprise 7 Server, Red Hat Enterprise 7 Workstation, or HPC Compute Node with one of the following commands, respectively.
   
   ```
   $ 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
   ```

6. Verify the list of subscriptions attached to your system.
   
   ```
   $ sudo subscription-manager list --consumed
   ```

7. Install the scl tool.
   
   ```
   $ sudo yum install scl-utils
   ```
1.2. INSTALL .NET CORE

1. Install .NET Core 2.0 and all of its dependencies.

   $ sudo yum install rh-dotnet20 -y

2. Enable the rh-dotnet20 Software Collection environment so you can run dotnet commands in the bash shell.

   NOTE

   This procedure installs the .NET Core 2.0 SDK. A .NET Core 2.1 SDK is also available, which you can install via `sudo yum install rh-dotnet20-dotnet-sdk-2.1`. It can be installed and used side-by-side with the 2.0 SDK. By default, the 2.1 SDK takes precedence if you install both. You can use a `global.json` to explicitly select one or the other.

   $ scl enable rh-dotnet20 bash

   NOTE

   This command does not persist; it creates a new shell, and the `dotnet` command is only available within that shell. If you log out, use another shell, or open up a new terminal, the `dotnet` command is no longer enabled.

   WARNING

   Red Hat does not recommend permanently enabling rh-dotnet20 because it may affect other programs. For example, rh-dotnet20 includes a version of libcurl that differs from the base RHEL version. This may lead to issues in programs that do not expect a different version of libcurl. If you want to permanently enable rh-dotnet for yourself, add the following line to your `~/.bashrc` file.

   source scl_source enable rh-dotnet20

3. Run the following command to prove the installation succeeded.

   $ dotnet --help
   .NET Command Line Tools (2.0.0)
   Usage: dotnet [runtime-options] [path-to-application]
   Usage: dotnet [sdk-options] [command] [arguments] [command-options]

   path-to-application:
   The path to an application .dll file to execute.

   SDK commands:
   new Initialize .NET projects.
1.3. CREATE AN APPLICATION

1. If you want to run the classic "Hello World" test case, create the following directory.

   $ mkdir hello-world

2. Navigate to the hello-world directory.

   $ cd hello-world

3. Create a project.

   $ dotnet new console
   The template "Console Application" was created successfully.
   Processing post-creation actions...
   Running 'dotnet restore' on /home/<USER>/hello-world/hello-world.csproj...
   Restore succeeded.

4. Pull the dependencies needed for the project.

   $ dotnet build

5. Run the project.

   $ dotnet run
   Hello World!

1.4. PUBLISH APPLICATIONS

The .NET Core 2.0 applications can be published to use a shared system-wide version of .NET Core or to include .NET Core. These two deployment types are called framework-dependent deployment (FDD) and self-contained deployment (SCD), respectively.

For RHEL, we recommend publishing by FDD. This method ensures the application is using an up-to-date version of .NET Core, built by Red Hat, that includes a specific set of native dependencies. These native libraries are part of the dotnet Software Collection. On the other hand, SCD uses a runtime built by Microsoft and uses the global Red Hat Enterprise Linux libraries. There are known issues when using .NET Core with these libraries.

1.4.1. Publish .NET Core Applications
1. Use the following command to publish an application using the FDD.

   $ dotnet publish -f netcoreapp2.0 -c Release

2. If the application will only be used on RHEL, you can trim the dependencies needed for other platforms by using these commands.

   $ dotnet restore -r rhel.7-x64
   $ dotnet publish -f netcoreapp2.0 -c Release -r rhel.7-x64 --self-contained false

3. Enable the Software Collection and pass the application assembly name to the dotnet command to run the application on a RHEL system.

   $ scl enable rh-dotnet20 -- dotnet <app>.dll

4. This command can be added to a script that is published with the application. Add the following script to your project and update the ASSEMBLY variable.

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

5. To include the script when publishing, add this ItemGroup to the csproj file.

   <ItemGroup>
   <None Update="<scriptname>">
   <ItemGroup>
   
   <PropertyGroup>
   <TargetFramework>netcoreapp2.0</TargetFramework>
   <PublishWithAspNetCoreTargetManifest>false</PublishWithAspNetCoreTargetManifest>
   </PropertyGroup>
   <ItemGroup>
   <PackageReference Include="Microsoft.AspNetCore.All">

1.4.2. Publish ASP.NET Core Applications

1. By default, ASP.NET Core 2.0 web applications are published with a dependency on a runtime store. This is a set of packages that are expected to be available on the runtime system. They are not included with the published application. RHEL does not include the ASP.NET runtime store, so the applications must be published, including all dependencies. This can be done by setting the PublishWithAspNetCoreTargetManifest property to false in the project file.

   <Project Sdk="Microsoft.NET.Sdk.Web">
   <PropertyGroup>
   "eval(s=SourceTransformSource("<targetframework>") + SourceTransformSource("<publishtype>" + SourceTransformSource("<config>")) + SourceTransformSource("<artifactfolder>"))
   </PropertyGroup>
   </Project>
2. Alternatively, this property can be set when publishing the application.

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

1.5. RUN APPLICATIONS ON DOCKER

This section shows how to use the dotnet/dotnet-20-runtime-rhel7 image to run your application inside a Docker container. It requires you to have the docker binary installed, the docker daemon running, and the rh-dotnet20 Software Collection enabled.

1. Create and navigate to the following directory.

   $ mkdir mvc_runtime_example && cd mvc_runtime_example

2. Create a new project.

   $ dotnet new mvc --no-restore

3. Pull all dependencies.

   $ dotnet restore -r rhel.7-x64

4. Publish the project.

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

5. Create a file called Dockerfile and add the following contents.

   $ cat > Dockerfile <<EOF
   FROM registry.access.redhat.com/dotnet/dotnet-20-runtime-rhel7
   ADD bin/Release/netcoreapp2.0/rhel.7-x64/publish/ .
   CMD ["dotnet", "mvc_runtime_example.dll"]
   EOF

6. Build your image.

   $ sudo docker build -t dotnet-20-runtime-example .

7. Run your image.

   $ sudo docker run -d -p8080:8080 dotnet-20-runtime-example

8. View the result in a browser: http://127.0.0.1:8080
Report a bug
2.1. INSTALL IMAGE STREAMS

The .NET Core image streams definition can be defined globally in the openshift namespace or locally in your specific project.

1. If you are a system administrator or otherwise have sufficient permissions, change to the openshift project. Using the openshift project allows you to globally update the image stream definitions.

   $ oc project openshift

   If you do not have permissions to use the openshift project, you can still update your project definitions starting with Step 2.

2. Run the following command to list all available .NET Core image versions.

   $ oc describe is dotnet

   The output shows installed images or the message Error from server (NotFound) if no images are installed.

3. To pull the images, OpenShift needs credentials for authenticating with the registry.redhat.io server. These credentials are stored in a secret.

   NOTE

   For OpenShift 3.11 and later, a secret is preconfigured for the openshift namespace.

   Enter the following command to list secrets. The first column shows the secret name.

   $ oc get secret | grep kubernetes.io/dockercfg

   To check the contents of a secret, you can decode the .dockercfg or .dockercfgjson data from Base64 format. This allows you to see if you already have credentials for the registry.redhat.io server. Enter the following command to show the .dockercfg section in a secret.

   $ oc get secret <secret-name> -o yaml | grep .dockercfg

   .dockercfg:
   eyJyZWdpc3RyeS5yZWRoYXQuaW8iOnsidXNlcm5hbWUiOiJkioqKioqKioqKioqKioqKioqKioqKioqKioqKiIsInBhc3N3
   b3JkIjoikioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKiIsInBhc3N3
   b3JkIjoikioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKiIsInBhc3N3
   b3JkIjoikioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKiIsInBhc3N3

   Copy and paste the output in the following command to convert it from Base64 format. The example below shows the credentials for the registry.redhat.io server.

   $ echo
   eyJyZWdpc3RyeS5yZWRoYXQuaW8iOnsidXNlcm5hbWUiOiJkioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKiIsInBhc3N3
   b3JkIjoikioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKiIsInBhc3N3
   b3JkIjoikioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKiIsInBhc3N3
   b3JkIjoikioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKiIsInBhc3N3
   b3JkIjoikioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKiIsInBhc3N3
   b3JkIjoikioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKiIsInBhc3N3
   b3JkIjoikioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKiIsInBhc3N3
You need to add a secret if there is no secret listed with credentials for the registry.redhat.io server.

4. Red Hat account credentials are used for registry.redhat.io access. If you are a customer with entitlements to Red Hat products, you already have account credentials to use. These are typically the same credentials used to log in to the Red Hat Customer Portal. To verify your Red Hat credentials, enter the following command and attempt to log in.

   $ docker login registry.redhat.io

If you cannot log in, you first need to get an account with Red Hat. See Red Hat Container Registry Authentication for additional information. If you can log in, enter the following commands to create the secret.

   $ oc create secret docker-registry redhat-registry 
   --docker-server=registry.redhat.io 
   --docker-username=<user-name> 
   --docker-password=<password> 
   --docker-email=unused

   $ oc secrets link default redhat-registry --for=pull
   $ oc secrets link builder redhat-registry

5. After creating the secret, enter the following command to import new image streams.

   $ oc create -f https://raw.githubusercontent.com/redhat-developer/s2i-dotnetcore/master/dotnet_imagestreams.json

If image streams were already installed, use the replace command to update the image stream definitions.

   $ oc replace -f https://raw.githubusercontent.com/redhat-developer/s2i-dotnetcore/master/dotnet_imagestreams.json

### 2.2. DEPLOY APPLICATIONS

1. Run the following commands to deploy the ASP.NET Core application, which is in the app folder on the dotnetcore-2.0 branch of the redhat-developer/s2i-dotnetcore-ex GitHub repository.

   $ oc new-app --name=exampleapp 'dotnet:2.0-https://github.com/redhat-developer/s2i-dotnetcore-ex#dotnetcore-2.0' --build-env DOTNET_STARTUP_PROJECT=app

2. As suggested by the output of the new-app command, you can track progress of the build using the oc logs command.

   $ oc logs -f bc/exampleapp
3. Once the build is finished, you can see the deployed application.

   $ oc logs -f dc/exampleapp

4. At this point, the application is accessible within the project. To make it accessible externally, use the `oc expose` command. You can then use `oc get routes` to find the URL.

   $ oc expose svc/exampleapp
   $ oc get routes

### 2.3. ENVIRONMENT VARIABLES

The .NET Core images support a number of environment variables to control the build behavior of your .NET Core application. These variables can be set as part of the build configuration, or they can be added to an `.s2i/environment` file in the application source code repository.

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOTNET_STARTUP_PROJECT</td>
<td>Used to select the project to run. This must be a project file (for example, csproj or fsproj) or a folder containing a single project file.</td>
<td>.</td>
</tr>
<tr>
<td>DOTNET_SDK_VERSION</td>
<td>Used to select the default sdk version when building. If there is a global.json file in the source repository, that takes precedence. When set to <code>latest</code> the latest sdk in the image is used.</td>
<td>Lowest sdk version available in the image</td>
</tr>
<tr>
<td>DOTNET_ASSEMBLY_NAME</td>
<td>Used to select the assembly to run. This must not include the .dll extension. Set this to the output assembly name specified in csproj (PropertyGroup/AssemblyName).</td>
<td>Name of the csproj file</td>
</tr>
<tr>
<td>DOTNET_RESTORE_SOURCES</td>
<td>Used to specify the space-separated list of NuGet package sources used during the restore operation. This overrides all of the sources specified in the NuGet.config file.</td>
<td></td>
</tr>
<tr>
<td>DOTNET_NPMTOOLS</td>
<td>Used to specify a list of NPM packages to install before building the application.</td>
<td></td>
</tr>
<tr>
<td>Variable Name</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>DOTNET_TEST_PROJECTS</td>
<td>Used to specify the list of test projects to test. This must be project files or folders containing a single project file. <code>dotnet test</code> is invoked for each item.</td>
<td></td>
</tr>
<tr>
<td>DOTNET_VERBOSITY</td>
<td>Used to specify the verbosity of the dotnet build commands. When set, the environment variables are printed at the start of the build. This variable can be set to one of the msbuild verbosity values, <code>q[uiet], m[inimal], n[ormal], d[etailed], and diag[nostic]</code>.</td>
<td></td>
</tr>
<tr>
<td>DOTNET_CONFIGURATION</td>
<td>Used to run the application in Debug or Release mode. This value should be either <code>Release</code> or <code>Debug</code>.</td>
<td>Release</td>
</tr>
<tr>
<td>ASPNETCORE_URLS</td>
<td>This variable is set to <code>http://*:8080</code> to configure ASP.NET Core to use the port exposed by the image. Changing this is not recommended.</td>
<td><code>http://*:8080</code></td>
</tr>
<tr>
<td>HTTP_PROXY, HTTPS_PROXY</td>
<td>Configures the HTTP/HTTPS proxy used when building and running the application.</td>
<td></td>
</tr>
<tr>
<td>NPM_MIRROR</td>
<td>Use a custom NPM registry mirror to download packages during the build process.</td>
<td></td>
</tr>
<tr>
<td>DOTNET_ASPNET_STORE</td>
<td>Publish assuming the runtime image contains the ASP.NET Core runtime store.</td>
<td><code>false</code></td>
</tr>
</tbody>
</table>

### 2.4. SAMPLE APPLICATIONS

Three sample applications are available:

- **dotnet-example**: This is the default model–view–controller (MVC) application.
- **dotnet-runtime-example**: This shows how to build an MVC application using a chained build and the .NET runtime image.
• **dotnet-pgsql-persistent**: This is the Microsoft ASP.NET Core MusicStore sample application using a PostgreSQL backend.

To add the samples using the OpenShift Web Console, browse to your project and click **Add to project**. You can filter for **dotnet**. If the samples do not show up, you can add them to your installation by running the following commands.

```sh
$ oc create -f https://raw.githubusercontent.com/redhat-developer/s2i-dotnetcore/master/templates/dotnet-example.json
$ oc create -f https://raw.githubusercontent.com/redhat-developer/s2i-dotnetcore/master/templates/dotnet-runtime-example.json
```

### 2.5. CREATE A RUNTIME IMAGE

The .NET Core runtime image contains the files sufficient to run a .NET application, but not to build one. As such, you must provide an already published application for the runtime image. This can be a static application inside of an image that layers on top of the runtime image or a new image can be built that is fed the application from a build image. This last method is commonly referred to as **chaining builds**.

Deploying a build chain requires a more in-depth configuration than a simple application build. At a minimum, a build chain needs:

- a build config for the build image
- a build config for the runtime image
- an inline dockerfile for the runtime image
- a deployment config that uses the runtime image

The following elements are optional but recommended:

- an internal image stream
- a trigger for the build image based on the source project
- a trigger for the runtime build based on the build image

See [dotnet-runtime-example template](#) for an example of such a configuration.

In this template, we define a *-build and a *-runtime build config. The build image builds from a GitHub project and has a trigger to rebuild automatically when the project changes or when the base image (in this case, **dotnet-20-rhel7**) updates.

The runtime image builds from the **dotnet-20-runtime-rhel7** base image but pulls a **tar.gz** file from the build image that contains the build project. That is done using this source definition in the build config.

```json
"source": {
   "dockerfile": "FROM ${DOTNET_RUNTIME_IMAGE_STREAM_TAG}\nADD app.tar.gz",
   "images": [
      { "from": {
```
The runtime image also has triggers to rebuild automatically when build image is updated in the projects internal image stream or when the `dotnet-20-runtime-rhel7` base image updates.

There is also a deployment defined for the runtime image. A deployment is not necessary for the build image.

The use of these triggers means that any object in the chain will be rebuilt as necessary when images or code is updated.

Report a bug
# APPENDIX A. REVISION HISTORY

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Author</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>08/21/2017</td>
<td>2.0</td>
<td>Les Williams</td>
<td>Generally available</td>
</tr>
<tr>
<td>08/30/2017</td>
<td>2.0</td>
<td>Les Williams</td>
<td>Revised DOTNET_STARTUP_PROJECT and DOTNET_TEST_PROJECT CTS entries in Section 2.3</td>
</tr>
<tr>
<td>09/13/2017</td>
<td>2.0</td>
<td>Les Williams</td>
<td>Revised Section 1.2 to include a note about how to permanently enable rh-dotnet20</td>
</tr>
<tr>
<td>02/14/2018</td>
<td>2.0</td>
<td>Les Williams</td>
<td>Revised Section 2.2 to resolve BZ 1500230; added quoting for zsh and other shells</td>
</tr>
<tr>
<td>02/28/2018</td>
<td>2.0.3</td>
<td>Les Williams</td>
<td>Revised to include SDK 2.0 and 2.1</td>
</tr>
<tr>
<td>06/07/2018</td>
<td>2.0.3</td>
<td>Toby Drake</td>
<td>Updated list of environment variables based on s2i-dotnetcore 2.0 build on github</td>
</tr>
<tr>
<td>08/27/2018</td>
<td>2.0.3</td>
<td>Toby Drake</td>
<td>Backported Install Stream section updates from .NET Core 2.1</td>
</tr>
</tbody>
</table>

[Report a bug]