.NET Core 1.1

Getting Started Guide

Installing .NET Core on Red Hat Enterprise Linux

Last Updated: 2018-06-04

.NET Core 1.1 Getting Started Guide

Installing .NET Core on Red Hat Enterprise Linux

Legal Notice

Abstract

The .NET Core platform is a general purpose development platform featuring automatic memory management and modern programming languages. It allows users to build high-quality applications efficiently. The .NET Core platform is available in Red Hat Enterprise Linux (RHEL 7) and OpenShift Container Platform via certified containers. The .NET Core platform 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. a heterogeneous datacenter, where the underlying infrastructure is capable of running .NET applications without having to rely solely on Windows Server. access to many of the popular development frameworks, such as .NET, Java, Ruby, and Python from within OpenShift Container Platform. .NET Core 1.1 is supported on Red Hat Enterprise Linux 7 and OpenShift Container Platform versions 3.3 and later.

Table of Contents

CHAPTER 1. INSTALL .NET CORE 1.1 ON RED HAT ENTERPRISE LINUX

This Getting Started Guide describes how to install .NET Core 1.1 on Red Hat Enterprise Linux (RHEL).

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.

   ```
   # subscription-manager register
   ```

3. Display a list of all subscriptions that are available for your system and identify the pool ID for the subscription.

   ```
   # subscription-manager list --available
   ```

   This command displays its 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. Replace *pool_id* with the pool ID you identified in the previous step.

   ```
   # subscription-manager attach --pool=<appropriate pool ID from the
   above step>
   ```

5. Verify the list of subscriptions attached to your system.

   ```
   # subscription-manager list --consumed
   ```

6. 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.

   ```
   # subscription-manager repos --enable=rhel-7-server-dotnet-rpms
   # subscription-manager repos --enable=rhel-7-workstation-dotnet-rpms
   # subscription-manager repos --enable=rhel-7-hpc-node-dotnet-rpms
   ```

7. Install the scl tool.

   ```
   # yum install scl-utils
   ```

1.1. INSTALL .NET CORE 1.1

1. Install .NET Core 1.1 and all of its dependencies.

```
# yum install rh-dotnetcore11
```

2. Enable the rh-dotnetcore11 collection environment.

```
$ scl enable rh-dotnetcore11 bash
```

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. Consider permanently enabling it by adding the following line to your **~/.bashrc** file.

```
source scl_source enable rh-dotnetcore11
```

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

```
$ dotnet --help
```

1.2. CREATE A .NET CORE 1.1 PROJECT

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 .NET Core 1.1 project.

```
$ dotnet new
```

4. Pull the dependencies needed for the .NET Core 1.1 project.

```
$ dotnet restore
```

5. Run the .NET Core 1.1 project.

```
$ dotnet run
```

Report a bug

CHAPTER 2. .NET CORE 1.1 ON RED HAT OPENSHIFT CONTAINER PLATFORM

Images are available for using .NET Core 1.1 on OpenShift Container Platform. See OpenShift Container Platform 3.3 Image Creation Guide for more details.

The .NET Core 1.1 software collection (rh-dotnetcore11) ships with the project.json build system (1.0.0-preview2 SDK). See the Known Issues in the .NET Core 1.1 Release Notes for details on installing this SDK on a non-RHEL system. Visual Studio 2017 dropped support for the project.json build system. Support for the msbuild/csproj build system will be added in the .NET Core 2.0 release.

2.1. IMAGES

Image stream definitions for the .NET Core on Red Hat Enterprise Linux Source to Image (S2I) are now added during OpenShift Container Platform installations. The RHEL 7 images are available through Red Hat's subscription registry using the following command.

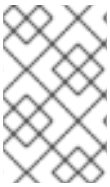
```
$ docker pull registry.access.redhat.com/dotnet/dotnetcore-11-rhel7
```

To use this image, you can either access it directly from image registries or push them into your OpenShift Container Platform Docker registry. You can also create an image stream that points to the image, either in your Docker registry or at the external location. Your OpenShift Container Platform resources can then reference the image stream definition. See OpenShift Container Platform 3.4 Guide to Using Images for more information about using images.

See OpenShift Container Platform 3.3 Installation and Configuration for more information about Red Hat OpenShift Container Platform 3.3 and .NET Core.

2.2. CONFIGURATION

The .NET Core images support a number of environment variables to control the build behavior of your .NET Core application.



NOTE

You must set environment variables that control build behavior in the s2i build configuration or in the **`.s2i/environment`** file to make them available to the build steps.

Variable Name	Description	Default
DOTNET_STARTUP_PROJECT	Used to select the project to run. This must be the folder containing **`project.json`**	Current working directory

Variable Name	Description	Default
DOTNET_PUBLISH	Used to control whether the application should be built by executing **`dotnet build`** or **`dotnet publish`**. To publish the application, set the value to *true*. It is recommended to publish your application.	For backwards compatibility, the default is *false*. In the next major release, this variable will be removed and the builder will always publish the application.
DOTNET_ASSEMBLY_NAME	Used to select the assembly to run. This must not include the **`.dll`** extension. Set this to the output assembly name specified in **`project.json`** (name, buildOptions/outputName). For **`project.json`**, the assembly name defaults to the **`project.json`** parent folder. When **`project.json`** is at the **`context-dir`**, the parent folder name will be **`src`**. So, by default, this generates a **`src.dll`** assembly. Setting **DOTNET_ASSEMBLY_NAME** will cause: * The assembly to be **`<DOTNET_ASSEMBLY_NAME>.dll`** * The application sources to be in subfolder **`DOTNET_ASSEMBLY_NAME`** in the deployed container.	The name of the **`DOTNET_STARTUP_PROJECT`** folder
DOTNET_RESTORE_SOURCES	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.	Unset
DOTNET_NPM_TOOLS	Used to specify a list of NPM packages to install before building the application	Unset

Variable Name	Description	Default
DOTNET_TEST_PROJECTS	Used to specify the space-separated list of test projects to run. This must be folders containing **`project.json`**. **`dotnet test`** is invoked for each folder.	Unset
DOTNET_CONFIGURATION	Used to run the application in Debug or Release mode. This value should be either *Release* or *Debug*.	**Release**
ASPNETCORE_URLS	This variable is set to **`http://*:8080`** to configure ASP.NET Core to use the port exposed by the image. It is not recommended to change this.	**`http://*:8080`**
HTTP_PROXY	Configures the HTTP proxy used when building and running the application	
HTTPS_PROXY	Configures the HTTPS proxy used when building and running the application	
NPM_MIRROR	Use a custom NPM registry mirror to download packages during the build process.	

NOTE

Typical modern web applications rely on javascript tools to build the front end. The image includes npm (node package manager) to install these tools. Packages can be installed by setting **DOTNET_NPM_TOOLS** and by calling **`npm install`** in the build process.

2.3. QUICKLY DEPLOY APPLICATIONS FROM .NET CORE SOURCE

An image can be used to build an application by running **`oc new-app`** against a sample repository.

```
$ oc new-app registry.access.redhat.com/dotnet/dotnetcore-11-
rhel7~https://github.com/redhat-developer/s2i-dotnetcore-ex#dotnetcore-1.1
--context-dir=app
```

Report a bug

CHAPTER 3. DOCKER

Docker is an open source project that extends Linux containers to provide the capability to package an application with its runtime dependencies. It provides a docker command-line interface (CLI) to manage container images.

The current releases of Red Hat Enterprise Linux and Red Hat Enterprise Linux Atomic include two different versions of Docker. You can choose from:

- **`docker`**: This package includes the version of docker that is the default for the current release of Red Hat Enterprise Linux. Install this package if you want a more stable version of docker that is compatible with the current versions of Kubernetes and OpenShift available with Red Hat Enterprise Linux.
- **`docker-latest`**: This package includes a later version of docker that you can use if you want to work with newer features of docker. This version is not compatible with the versions of Kubernetes and OpenShift that are available with the current release of Red Hat Enterprise Linux.

NOTE

The docker, source-to-image packages, and running .NET Core container images are supported only on Red Hat Enterprise Linux 7 Server and Red Hat Enterprise Linux Atomic Host. You cannot install docker or run the images on Red Hat Enterprise Linux 7 Workstation or Red Hat Enterprise Linux 6 or earlier.

3.1. LINUX CONTAINERS

Linux Containers is a dense application packaging and isolation technology that provides resource management, process isolation, and security on a single host. Applications are packaged with their required runtime components and deployed on a certified Red Hat Enterprise Linux host. It allows one system to run multiple secure, isolated runtimes for applications to increase system utilization.

Several components are needed for Linux containers to function correctly, and most of them are provided by the Linux kernel. Kernel namespaces ensure process isolation, and Control Groups (cgroups) are employed to control the system resources. Security-Enabled Linux (SELinux) is used to assure separation between the host and the container and also between the individual containers. Management interface forms a higher layer that interacts with the aforementioned kernel components and provides tools for construction and management of containers.

Image-based containers allow you to host multiple instances and versions of an application with minimal overhead and increased flexibility. Such containers are not tied to the host-specific configuration, which makes them portable.

When using SELinux for controlling processes within a container, make sure that any content that is volume mounted into the container is readable and potentially writable, depending on the use case. For more information, see Using Volumes with the docker Container Can Cause Problems with SELinux.

For more information on containers and container images, see the Core Concepts of the OpenShift Enterprise 3.0 Architecture, which discusses core concepts and methods related to delivering containerized applications.

3.2. GET DOCKER IN RED HAT ENTERPRISE LINUX 7

To get an environment where you can develop Docker-formatted containers, you can install a RHEL 7 system to act as a development system as well as a container host. The docker package is stored in a Red Hat Enterprise Linux Extras repository. See the Red Hat Enterprise Linux Extras Life Cycle article for a description of support policies and life cycle information for the Red Hat Enterprise Linux Extras channel.

If you want to create Docker-formatted images or containers when using the Red Hat Enterprise Linux 7 subscription model, you must properly register and entitle the host computer on which you build them. When you use **`yum install`** within a container to add packages, the container automatically has access to entitlements available from the Red Hat Enterprise Linux 7 host so it can get RPM packages from any repository enabled on that host.

The Red Hat Enterprise Linux 7 base image container in docker-format can be found here.

NOTE

Currently, you must have root privilege to run the **`docker`** command in Red Hat Enterprise Linux 7 and Red Hat Enterprise Linux Atomic Host. Configuring sudo will work if you prefer not to log in directly to the root user account.

Section 1.3, Getting Docker in RHEL 7 describes how to install and register Red Hat Enterprise Linux 7 and then install docker.

3.3. GET DOCKER IN RED HAT ENTERPRISE LINUX 7 ATOMIC HOST

Because Red Hat Enterprise Linux Atomic Host is more like an appliance than a full-featured Linux system, it is not made for you to install RPM packages or other software on. Software is added to Red Hat Enterprise Linux Atomic Host systems by running container images.

Red Hat Enterprise Linux Atomic Host has a mechanism for updating existing packages, but it does not have a mechanism for allowing users to add new packages. You should consider using a standard Red Hat Enterprise Linux 7 Server system to develop your applications so you can add a full complement of development and debugging tools. You can then use Red Hat Enterprise Linux Atomic Host to deploy your containers into a variety of virtualization and cloud environments.

You can use a Red Hat Enterprise Linux Atomic Host system to run, build, stop, start, and otherwise work with containers. Section 1.4, Getting Docker in RHEL 7 Atomic Host describes how to install and register RHEL 7 and then install Docker.

3.4. WORKING WITH DOCKER-FORMATTED CONTAINERS

You can manage Docker-formatted images that are on your system in several ways, whether or not they have been run. The **`docker run`** command lets you say which command to run in a container. Once a container is running, you can stop, start, and restart it. You can remove containers you no longer need. (In fact, you probably want to.) Before you run an image, it is a good idea to investigate its contents.

See Section 1.7.5, Working with docker-formatted Containers for details on how to work with these containers.

3.5. USE .NET CORE CONTAINER IMAGES

There are two basic approaches that you can take to use the container images shipped with .NET Core:

- using base images
- using Source-to-Image (S2I).

3.5.1. Use Base Images

The Red Hat Enterprise Linux 7 images are available through Red Hat's subscription registry using the following command.

```
# docker pull registry.access.redhat.com/dotnet/dotnetcore-11-rhel7
```

To use container images provided by Red Hat as base images in your own Dockerfile, add the following line to it.

```
FROM registry.access.redhat.com/dotnet/dotnetcore-11-rhel7
```

Details about working with Dockerfiles is covered in Red Hat Enterprise Linux Atomic Host 7 Getting Started with Containers.

Detailed information on Dockerfiles can be found in the Dockerfile reference document.

The following steps will guide you through creating, building, publishing and deploying a .NET Core 1.1 project as a docker image on a Red Hat Enterprise Linux 7 Server:

1. Create a **`Dockerfile`** with the following contents:

   ```
   FROM registry.access.redhat.com/dotnet/dotnetcore-11-rhel7
   COPY hello-world /opt/app-root/src
   CMD dotnet bin/Release/netcoreapp1.1/publish/hello-world.dll
   ```

2. Create a hello-world project and publish it.

   ```
   $ scl enable rh-dotnetcore11 bash
   $ mkdir hello-world
   $ dotnet new && dotnet restore && dotnet publish c Release -r
   rhel.7.2-x64
   $ cd ..
   ```

3. Build the docker image and run the .NET Core 1.1 application inside it. You should see a "Hello World" message.

   ```
   # docker build -t dotnet-hello-world .
   # docker run dotnet-hello-world
   ```

3.5.2. Use Source-to-Image

Source-to-Image (S2I) is a framework and a tool for writing images that use the application source code as an input and produces a new image that runs the assembled application as an output. The main advantage of using the S2I tool for building reproducible container

images is the ease of use for developers.

See Section 2.2, “Configuration” for details about using environment variables to control the build behavior of your .NET Core application.

1. To use the S2I tool on your system, ensure that the *rhel-7-server-extras-rpms* channel is enabled.

```
# subscription-manager repos --enable=rhel-7-server-extras-rpms
```

2. Run the following command to install the S2I package.

```
# yum install source-to-image
```

The build process consists of the three fundamental elements that are combined into a final container image:

- Source code of your .NET Core application
- Builder image, a container image provided by Red Hat that supports building images using the S2I tool
- S2I scripts that are part of the builder image.

During the build process, S2I creates a tar file that contains the source code and scripts, and then it streams that file into the builder image.

For more information on the Source-to-Image framework, see S2I Requirements.

More information about the S2I tool is available at GitHub.

1. Build the test application from the .NET Core repository on GitHub, underneath the **`1.1/test/asp-net-hello-world/`** directory.

```
$ sudo s2i build git://github.com/redhat-developer/s2i-dotnetcore --
context-dir=1.1/test/asp-net-hello-world dotnet/dotnetcore-11-rhel7
dotnetcore11-rhel7-app
```

   This produces a new application image: dotnetcore-11-rhel7-app.

2. Run the resulting dotnetcore-11-rhel7-app image:

```
# docker run -d -p 8080-8081:8080-8081 --name example-app
dotnetcore-11-rhel7-app
```

3. See the app running at http://localhost:8080/.

```
$ curl http://localhost:8080/
```

4. "Hello World" is returned.

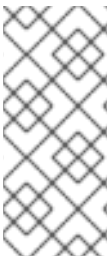
5. See the app running at https://localhost:8081/.

```
$ curl -1 --insecure https://localhost:8081/
```

6. Hello World is returned.

7. Stop the container.

```
# docker stop example-app
```



NOTE

See Chapter 12 in Transitioning to .NET Core on Red Hat Enterprise Linuxfor details on how to build, configure, and run a .NET Core image in a container. The instructions describe how to do this in .NET Core 1.0, but you should be able to run the same commands by replacing 1.0 and 10 with 1.1 and 11, respectively, if you are using .NET Core 1.1 instead of .NET Core 1.0.

Report a bug

CHAPTER 4. .NET CORE TEMPLATES AND SAMPLE APPS

4.1. .NET CORE TEMPLATES

The .NET Core image templates and the .NET Core images streams must first be installed. The new templates are:

- dotnet-example
- dotnet-pgsql-persistent

Use this command to check that the image templates are installed.

```
$ (oc get -n openshift templates; oc get -n openshift is) | grep dotnet
```

4.2. .NET CORE SAMPLE APPS

We recommend using redhat-developer/s2i-dotnetcore-ex as the primary sample application. The new sample app provides a more exciting example than the simple "Hello World" app.

See the .NET Core repo on GitHub for more sample apps.

1. Deploy the .NET Core sample application running on dotnet/dotnetcore-11-rhel7 using the dotnet-example template with the following command.

```
$ oc new-app --template dotnet-example \
-p DOTNET_IMAGE_STREAM_TAG=dotnet:1.1 \
-p SOURCE_REPOSITORY_REF=dotnetcore-1.1
```

2. Deploy the .NET Core sample application using PostgreSQL as database with the following command.

```
$ oc new-app --template=dotnet-pgsql-persistent
```

Report a bug

CHAPTER 5. REFERENCE MATERIAL

Multiple implementations of .NET Core are available, based on open .NET Standards that specify the fundamentals of the platform. See .NET Standards for more information about the standards and the Common Language Infrastructure.

See The Book of Runtime and the CoreFX framework for more information about the various Common Language Runtime libraries and framework.

See Red Hat Enterprise Linux documentation for more information about Red Hat Enterprise Linux 7.

Report a bug

APPENDIX A. REVISION HISTORY

Date	Version	Author	Changes
06/24/2016	1.0	Les Williams	Original version
07/27/2016	1.0	Les Williams	Revised version number to reflect top-level version and spelled out RHEL
08/29/2016	1.0	Les Williams	Removed Step 8 and removed command outputs from Steps 9, 11, 14, 15, and 16
09/23/2016	1.0	Les Williams	Revised the link for Common Language Runtime (CLR) and added a link for Common Language Infrastructure (CLI)
11/7/2016	1.0	Les Williams	Added references to Red Hat Enterprise Linux 7 Workstation and added a link for permanently enabling rh-dotnetcore10 bash
11/29/2016	1.1	Les Williams	First minor release
12/09/2016	1.1	Les Williams	Revised support information
12/20/2016	1.1	Les Williams	Revised Section 3.5.2 to ensure the rhel-7-server-extras-rpms channel is enabled before invoking "yum install source-to-image"
03/20/2017	1.1.1	Les Williams	Revised to include environment variables
04/04/2017	1.1.1	Les Williams	Revised to include container templates

Date	Version	Author	Changes
04/26/2017	1.1.1	Les Williams	Revised to include build system support information
05/16/2017	1.1.2	Les Williams	Revised to include three new environment variables

Report a bug