



.NET 3.1

Getting started with .NET on RHEL 7

Installing and running .NET Core 3.1 on RHEL 7

.NET 3.1 Getting started with .NET on RHEL 7

Installing and running .NET Core 3.1 on RHEL 7

Legal Notice

Copyright © 2022 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux[®] is the registered trademark of Linus Torvalds in the United States and other countries.

Java[®] is a registered trademark of Oracle and/or its affiliates.

XFS[®] is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL[®] is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js[®] is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack[®] Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

Abstract

This guide describes how to install and run .NET Core 3.1 on RHEL 7.

Table of Contents

MAKING OPEN SOURCE MORE INCLUSIVE	3
PROVIDING FEEDBACK ON RED HAT DOCUMENTATION	4
CHAPTER 1. INTRODUCING .NET CORE 3.1	5
CHAPTER 2. INSTALLING .NET CORE 3.1	6
CHAPTER 3. CREATING AN APPLICATION USING .NET CORE 3.1	8
CHAPTER 4. PUBLISHING APPLICATIONS WITH .NET CORE 3.1	9
4.1. PUBLISHING .NET CORE APPLICATIONS	9
CHAPTER 5. RUNNING .NET CORE 3.1 APPLICATIONS IN CONTAINERS	11
CHAPTER 6. USING .NET CORE 3.1 ON OPENSIFT CONTAINER PLATFORM	13
6.1. INSTALLING .NET CORE IMAGE STREAMS	13
6.1.1. Installing image streams using OpenShift Client	13
6.1.2. Installing image streams on Linux and macOS	13
6.1.3. Installing image streams on Windows	14
6.2. DEPLOYING APPLICATIONS FROM SOURCE USING OC	15
6.3. DEPLOYING APPLICATIONS FROM BINARY ARTIFACTS USING OC	16
6.4. USING JENKINS SLAVE	16
6.5. ENVIRONMENTAL VARIABLES FOR .NET CORE 3.1	18
6.6. CREATING THE MVC SAMPLE APPLICATION	21
6.7. CREATING THE CRUD SAMPLE APPLICATION	21
CHAPTER 7. MIGRATION FROM PREVIOUS VERSIONS OF .NET	23
7.1. MIGRATION FROM PREVIOUS VERSIONS OF .NET	23
7.2. PORTING FROM .NET FRAMEWORK	23

MAKING OPEN SOURCE MORE INCLUSIVE

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see [our CTO Chris Wright's message](#).

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

We appreciate your input on our documentation. Please let us know how we could make it better. To do so:

- For simple comments on specific passages:
 1. Make sure you are viewing the documentation in the *Multi-page HTML* format. In addition, ensure you see the **Feedback** button in the upper right corner of the document.
 2. Use your mouse cursor to highlight the part of text that you want to comment on.
 3. Click the **Add Feedback** pop-up that appears below the highlighted text.
 4. Follow the displayed instructions.
- For submitting more complex feedback, create a Bugzilla ticket:
 1. Go to the [Bugzilla](#) website.
 2. As the Component, use **Documentation**.
 3. Fill in the **Description** field with your suggestion for improvement. Include a link to the relevant part(s) of documentation.
 4. Click **Submit Bug**.

CHAPTER 1. INTRODUCING .NET CORE 3.1

.NET Core is a general-purpose development platform featuring automatic memory management and modern programming languages. Using .NET, you can build high-quality applications efficiently. .NET Core is available on Red Hat Enterprise Linux (RHEL) and OpenShift Container Platform through certified containers.

.NET Core offers the following features:

- 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 on RHEL and OpenShift Container Platform.
- The capacity to more easily develop new .NET Core workloads on Microsoft Windows. You can deploy and run your applications on either RHEL or Windows Server.
- A heterogeneous data center, where the underlying infrastructure is capable of running .NET applications without having to rely solely on Windows Server.

.NET Core 3.1 is supported on RHEL 7, RHEL 8, RHEL 9, and OpenShift Container Platform versions 3.3 and later.

CHAPTER 2. INSTALLING .NET CORE 3.1

To install .NET Core on RHEL 7 you need to first enable the .NET Core software repositories and install the **scl** tool.

Prerequisites

- Installed and registered RHEL 7 with attached subscriptions.
For more information, see [Registering the System and Attaching Subscriptions](#).

Procedure

1. Enable the .NET Core software repositories:

```
$ sudo subscription-manager repos --enable=rhel-7-variant-dotnet-rpms
```

Replace *variant* with **server**, **workstation** or **hpc-node** depending on what RHEL system you are running (RHEL 7 Server, RHEL 7 Workstation, or HPC Compute Node, respectively).

2. Verify the list of subscriptions attached to your system:

```
$ sudo subscription-manager list --consumed
```

3. Install the **scl** tool:

```
$ sudo yum install scl-utils -y
```

4. Install .NET Core 3.1 and all of its dependencies:

```
$ sudo yum install rh-dotnet31 -y
```

5. Enable the **rh-dotnet31** Software Collection environment:

```
$ scl enable rh-dotnet31 bash
```

You can now run **dotnet** commands in this **bash** shell session.

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-dotnet31** because it may affect other programs. For example, **rh-dotnet31** 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 enable **rh-dotnet** permanently, add **source scl_source enable rh-dotnet31** to your **~/.bashrc** file.

Verification steps

- Verify the installation:

```
$ dotnet --info
```

The output returns the relevant information about the .NET Core installation and the environment.

CHAPTER 3. CREATING AN APPLICATION USING .NET CORE

3.1

Learn how to create a C# **hello-world** application.

Procedure

1. Create a new Console application in a directory called **my-app**:

```
$ dotnet new console --output my-app
```

The output returns:

```
The template "Console Application" was created successfully.
```

```
Processing post-creation actions...
```

```
Running 'dotnet restore' on my-app/my-app.csproj...
```

```
Determining projects to restore...
```

```
Restored /home/username/my-app/my-app.csproj (in 67 ms).
```

```
Restore succeeded.
```

A simple **Hello World** console application is created from a template. The application is stored in the specified **my-app** directory.

Verification steps

- Run the project:

```
$ dotnet run --project my-app
```

The output returns:

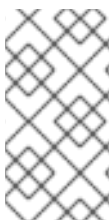
```
Hello World!
```

CHAPTER 4. PUBLISHING APPLICATIONS WITH .NET CORE 3.1

.NET Core 3.1 applications can be published to use a shared system-wide version of .NET Core or to include .NET Core.

The following methods exist for publishing .NET Core 3.1 applications:

- Framework-dependent deployment (FDD) - The application uses a shared system-wide version of .NET.



NOTE

When publishing an application for RHEL, Red Hat recommends using FDD, because it ensures that the application is using an up-to-date version of .NET Core, built by Red Hat, that uses a set of native dependencies. These native libraries are part of the **rh-dotnet31** Software Collection.

- Self-contained deployment (SCD) - The application includes .NET. This method uses a runtime built by Microsoft. Running applications outside the **rh-dotnet31** Software Collection may cause issues due to the unavailability of native libraries.

Prerequisites

- Existing .NET Core application.
For more information on how to create a .NET Core application, see

4.1. PUBLISHING .NET CORE APPLICATIONS

The following procedure outlines how to publish a framework-dependent application.

Procedure

1. Publish the framework-dependent application:

```
$ dotnet publish my-app -f netcoreapp3.1 -c Release
```

Replace *my-app* with the name of the application you want to publish.

2. **Optional:** If the application is for RHEL only, trim out the dependencies needed for other platforms:

```
$ dotnet restore my-app -r rhel.7-x64
$ dotnet publish my-app -f netcoreapp3.1 -c Release -r rhel.7-x64 --self-contained false
```

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

```
$ scl enable rh-dotnet31 -- dotnet <app>.dll
```

4. You can add the **scl enable rh-dotnet31 — dotnet <app>.dll** command to a script that is published with the application.
Add the following script to your project and update the **APP** variable:

```
#!/bin/bash

APP=<app>
SCL=rh-dotnet31
DIR="$(dirname "$(readlink -f "$0")")"

scl enable $SCL -- "$DIR/$APP" "$@"
```

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

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

CHAPTER 5. RUNNING .NET CORE 3.1 APPLICATIONS IN CONTAINERS

Use the **dotnet/dotnet-31-runtime-rhel7** image to run a precompiled application inside a Linux container.

Prerequisites

- Preconfigured containers.
The following example uses podman.

Procedure

1. **Optional:** If you are in another project's directory and do not wish to create a nested project, return to the parent directory of the project:

```
$ cd ..
```

2. Create a new MVC project in a directory called **mvc_runtime_example**:

```
$ dotnet new mvc --output mvc_runtime_example
```

3. Publish the project:

```
$ dotnet publish mvc_runtime_example -f netcoreapp3.1 -c Release
```

4. Create the **Dockerfile**:

```
$ cat > Dockerfile <<EOF
FROM registry.redhat.io/dotnet/dotnet-31-runtime-rhel7

ADD bin/Release/netcoreapp3.1/publish/ .

CMD ["dotnet", "mvc_runtime_example.dll"]
EOF
```

5. Build your image:

```
$ podman build -t dotnet-31-runtime-example .
```



NOTE

If you get an error containing the message **unable to retrieve auth token: invalid username/password**, you need to provide credentials for the **registry.redhat.io** server. Use the command **podman login registry.redhat.io** to log in. Your credentials are typically the same as those used for the Red Hat Customer Portal.

6. Run your image:

```
$ podman run -d -p8080:8080 dotnet-31-runtime-example
```

Verification steps

- View the application running in the container:

```
$ xdg-open http://127.0.0.1:8080
```

CHAPTER 6. USING .NET CORE 3.1 ON OPENSIFT CONTAINER PLATFORM

6.1. INSTALLING .NET CORE IMAGE STREAMS

To install .NET Core image streams, use image stream definitions from [s2i-dotnetcore](#) with the OpenShift Client (**oc**) binary. Image streams can be installed from Linux, Mac, and Windows. A script enables you to install, update or remove the image streams.

You can define .NET Core image streams in the global **openshift** namespace or locally in a project namespace. Sufficient permissions are required to update the **openshift** namespace definitions.

Obtaining the RHEL 7 image streams requires authentication against the **registry.redhat.io** server using subscription credentials. These credentials are configured by adding a pull secret to the OpenShift namespace.

6.1.1. Installing image streams using OpenShift Client

You can use OpenShift Client (**oc**) to install .NET Core image streams.

Prerequisites

- An existing pull secret must be present in the namespace. If no pull secret is present in the namespace. Add one by following the instructions in the [Red Hat Container Registry Authentication](#) guide.

Procedure

1. List the available .NET Core image streams:

```
$ oc describe is dotnet
```

The output shows installed images. If no images are installed, the **Error from server (NotFound)** message is displayed.

2. Install the .NET Core image streams:

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

3. When .NET Core image streams are already installed, you can include newer versions by running:

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

6.1.2. Installing image streams on Linux and macOS

You can use [this script](#) to install, upgrade, or remove the image streams on Linux and macOS.

Procedure

1. Download the script.

- a. On Linux use:

```
$ wget https://raw.githubusercontent.com/redhat-developer/s2i-dotnetcore/master/install-imagestreams.sh
```

- b. On Mac use:

```
$ curl https://raw.githubusercontent.com/redhat-developer/s2i-dotnetcore/master/install-imagestreams.sh -o install-imagestreams.sh
```

2. Make the script executable:

```
$ chmod +x install-imagestreams.sh
```

3. Log in to the OpenShift cluster:

```
$ oc login
```

4. Install image streams and add a pull secret for authentication against the **registry.redhat.io**:

```
./install-imagestreams.sh --os rhel7 [--user subscription_username --password subscription_password]
```

Replace *subscription_username* with the name of the user, and replace *subscription_password* with the user's password. The credentials may be omitted if you do not plan to use the RHEL7-based images.

If the pull secret is already present, the **--user** and **--password** arguments are ignored.

Additional information

- **./install-imagestreams.sh --help**

6.1.3. Installing image streams on Windows

You can use [this script](#) to install, upgrade, or remove the image streams on Windows.

Procedure

1. Download the script.

```
Invoke-WebRequest https://raw.githubusercontent.com/redhat-developer/s2i-dotnetcore/master/install-imagestreams.ps1 -UseBasicParsing -OutFile install-imagestreams.ps1
```

2. Log in to the OpenShift cluster:

```
$ oc login
```

3. Install image streams and add a pull secret for authentication against the **registry.redhat.io**:

```
./install-imagestreams.sh --OS rhel7 [-User subscription_username -Password
subscription_password]
```

Replace *subscription_username* with the name of the user, and replace *subscription_password* with the user's password. The credentials may be omitted if you do not plan to use the RHEL7-based images.

If the pull secret is already present, the **-User** and **-Password** arguments are ignored.



NOTE

The PowerShell **ExecutionPolicy** may prohibit executing this script. To relax the policy, run **Set-ExecutionPolicy -Scope Process -ExecutionPolicy Bypass -Force**.

Additional information

- **Get-Help .\install-imagestreams.ps1**

6.2. DEPLOYING APPLICATIONS FROM SOURCE USING **oc**

The following example demonstrates how to deploy the *example-app* application using **oc**, which is in the **app** folder on the **dotnetcore-3.1** branch of the **redhat-developer/s2i-dotnetcore-ex** GitHub repository:

Procedure

1. Create a new OpenShift project:

```
$ oc new-project sample-project
```

2. Add the ASP.NET Core application:

```
$ oc new-app --name=example-app 'dotnet:3.1~https://github.com/redhat-developer/s2i-dotnetcore-ex#dotnetcore-3.1' --build-env DOTNET_STARTUP_PROJECT=app
```

3. Track the progress of the build:

```
$ oc logs -f bc/example-app
```

4. View the deployed application once the build is finished:

```
$ oc logs -f dc/example-app
```

The application is now accessible within the project.

5. **Optional:** Make the project accessible externally:

```
$ oc expose svc/example-app
```

6. Obtain the shareable URL:

```
$ oc get routes
```

6.3. DEPLOYING APPLICATIONS FROM BINARY ARTIFACTS USING `oc`

You can use .NET Core Source-to-Image (S2I) builder image to build applications using binary artifacts that you provide.

Prerequisites

1. Published application.
For more information, see

Procedure

1. Create a new binary build:

```
$ oc new-build --name=my-web-app dotnet:3.1 --binary=true
```

2. Start the build and specify the path to the binary artifacts on your local machine:

```
$ oc start-build my-web-app --from-dir=bin/Release/netcoreapp3.1/publish
```

3. Create a new application:

```
$ oc new-app my-web-app
```

6.4. USING JENKINS SLAVE

The OpenShift Container Platform Jenkins image provides auto-discovery of the .NET Core 3.1 slave image (**dotnet-31**).

For auto-discovery to work, you need to add a Jenkins slave **ConfigMap** yaml file to the project.

Procedure

1. Change to the project where Jenkins is (or will be) deployed:

```
$ oc project _project-name_
```

2. Create a **dotnet-jenkins-slave.yaml** file:



NOTE

The value used for the **<serviceAccount>** element is the account used by the Jenkins slave. If no value is specified, the **default** service account is used.

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: dotnet-jenkins-slave-31
labels:
  role: jenkins-slave
data:
  dotnet31: |-
```

```

<org.csanchez.jenkins.plugins.kubernetes.PodTemplate>
  <inheritFrom></inheritFrom>
  <name>dotnet-31</name>
  <instanceCap>2247483647</instanceCap>
  <idleMinutes>0</idleMinutes>
  <label>dotnet-31</label>
  <serviceAccount>jenkins</serviceAccount>
  <nodeSelector></nodeSelector>
  <volumes/>
  <containers>
    <org.csanchez.jenkins.plugins.kubernetes.ContainerTemplate>
      <name>jnlp</name>
      <image>registry.access.redhat.com/dotnet/dotnet-31-jenkins-slave-
rhel7:latest</image>
      <privileged>>false</privileged>
      <alwaysPullImage>true</alwaysPullImage>
      <workingDir>/tmp</workingDir>
      <command></command>
      <args>${computer.jnlpMac} ${computer.name}</args>
      <ttyEnabled>>false</ttyEnabled>
      <resourceRequestCpu></resourceRequestCpu>
      <resourceRequestMemory></resourceRequestMemory>
      <resourceLimitCpu></resourceLimitCpu>
      <resourceLimitMemory></resourceLimitMemory>
      <envVars/>
    </org.csanchez.jenkins.plugins.kubernetes.ContainerTemplate>
  </containers>
  <envVars/>
  <annotations/>
  <imagePullSecrets/>
  <nodeProperties/>
</org.csanchez.jenkins.plugins.kubernetes.PodTemplate>

```

3. Import the configuration into the project:

```
$ oc create -f dotnet-jenkins-slave.yaml
```

The slave image can now be used.

Example

The following example shows a Jenkins pipeline added to OpenShift Container Platform. Note that when a Jenkins pipeline is added and no Jenkins master is running, OpenShift automatically deploys a master. See [OpenShift Container Platform and Jenkins](#) for additional information about deploying and configuring a Jenkins server instance.

In the example steps, the **BuildConfig** yaml file includes a simple Jenkins pipeline configured using the **dotnet-31** Jenkins slave. There are three stages in the example **BuildConfig** yaml file:

1. The sources are checked out.
2. The application is published.
3. The image is assembled using a binary build.
See [Section 6.3, "Deploying applications from binary artifacts using oc"](#) for additional information about binary builds.

Procedure

To configure the Jenkins master-slave pipeline:

1. Create the **buildconfig.yaml** file:

```
kind: BuildConfig
apiVersion: v1
metadata:
  name: dotnetapp-build
spec:
  strategy:
    type: JenkinsPipeline
    jenkinsPipelineStrategy:
      jenkinsfile: |-
        node("dotnet-31") {
          stage('clone sources') {
            sh "git clone https://github.com/redhat-developer/s2i-dotnetcore-ex --branch
dotnetcore-3.1 ."
          }
          stage('publish') {
            dir('app') {
              sh "dotnet publish -c Release"
            }
          }
          stage('create image') {
            dir('app') {
              sh 'oc new-build --name=dotnetapp dotnet:3.1 --binary=true || true'
              sh 'oc start-build dotnetapp --from-dir=bin/Release/{buildconfig-var}3.1/publish --
follow'
            }
          }
        }
      follow'
```

2. Import the **BuildConfig** file to OpenShift:

```
$ oc create -f buildconfig.yaml
```

3. Open the OpenShift console.
4. Go to **Builds → Pipelines**.
The **dotnetapp-build** pipeline is available.

5. Click **Start Pipeline**.

Note that it may take a while for the build to start because the Jenkins image(s) need to be downloaded first.

During the build you can watch the different pipeline stages complete in the OpenShift console. You can also click **View Log** to see the pipeline stages complete in Jenkins.

6. When the Jenkins pipeline build is complete, go to **Builds → Images**.
The **dotnetapp** image is built and available.

6.5. ENVIRONMENTAL VARIABLES FOR .NET CORE 3.1

The .NET Core images support several environment variables to control the build behavior of your .NET Core application. You can set these variables as part of the build configuration, or add them to the **.s2i/environment** file in the application source code repository.

Variable Name	Description	Default
DOTNET_STARTUP_PROJECT	Selects the project to run. This must be a project file (for example, csproj or fsproj) or a folder containing a single project file.	.
DOTNET_ASSEMBLY_NAME	Selects the assembly to run. This must not include the .dll extension. Set this to the output assembly name specified in csproj (PropertyGroup/AssemblyName).	The name of the csproj file
DOTNET_PUBLISH_READRYTORUN	When set to true , the application will be compiled ahead of time. This reduces startup time by reducing the amount of work the JIT needs to perform when the application is loading.	false
DOTNET_RESTORE_SOURCES	Specifies 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. This variable cannot be combined with DOTNET_RESTORE_CONFIGFILE .	
DOTNET_RESTORE_CONFIGFILE	Specifies a NuGet.Config file to be used for restore operations. This variable cannot be combined with DOTNET_RESTORE_SOURCES .	
DOTNET_TOOLS	Specifies a list of .NET tools to install before building the app. It is possible to install a specific version by post pending the package name with @<version> .	
DOTNET_NPM_TOOLS	Specifies a list of NPM packages to install before building the application.	

Variable Name	Description	Default
DOTNET_TEST_PROJECTS	Specifies the list of test projects to test. This must be project files or folders containing a single project file. dotnet test is invoked for each item.	
DOTNET_CONFIGURATION	Runs the application in Debug or Release mode. This value should be either Release or Debug .	Release
DOTNET_VERBOSITY	Specifies 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 (q[uiet] , m[inimal] , n[ormal] , d[etailed] , and diag[nostic]).	
HTTP_PROXY, HTTPS_PROXY	Configures the HTTP or HTTPS proxy used when building and running the application, respectively.	
DOTNET_RM_SRC	When set to true , the source code will not be included in the image.	
DOTNET_SSL_DIRS	Specifies a list of folders or files with additional SSL certificates to trust. The certificates are trusted by each process that runs during the build and all processes that run in the image after the build (including the application that was built). The items can be absolute paths (starting with /) or paths in the source repository (for example, certificates).	
NPM_MIRROR	Uses a custom NPM registry mirror to download packages during the build process.	
ASPNETCORE_URLS	This variable is set to http://*:8080 to configure ASP.NET Core to use the port exposed by the image. Changing this is not recommended.	http://*:8080

Variable Name	Description	Default
DOTNET_RESTORE_DISABLE_PARALLEL	When set to true , disables restoring multiple projects in parallel. This reduces restore timeout errors when the build container is running with low CPU limits.	false
DOTNET_INCREMENTAL	When set to true , the NuGet packages will be kept so they can be re-used for an incremental build.	false
DOTNET_PACK	When set to true , creates a tar.gz file at /opt/app-root/app.tar.gz that contains the published application.	

6.6. CREATING THE MVC SAMPLE APPLICATION

s2i-dotnetcore-ex is the default Model, View, Controller (MVC) template application for .NET Core.

This application is used as the example application by the .NET Core S2I image and can be created directly from the OpenShift UI using the *Try Example* link.

The application can also be created with the OpenShift client binary (**oc**).

Procedure

To create the sample application using **oc**:

1. Add the .NET Core application:

```
$ oc new-app dotnet:3.1~https://github.com/redhat-developer/s2i-dotnetcore-ex#dotnetcore-3.1 --context-dir=app
```

2. Make the application accessible externally:

```
$ oc expose service s2i-dotnetcore-ex
```

3. Obtain the sharable URL:

```
$ oc get route s2i-dotnetcore-ex
```

Additional resources

- [s2i-dotnetcore-ex application repository on GitHub](#)

6.7. CREATING THE CRUD SAMPLE APPLICATION

s2i-dotnetcore-persistent-ex is a simple Create, Read, Update, Delete (CRUD) .NET Core web application that stores data in a PostgreSQL database.

Procedure

To create the sample application using **oc**:

1. Add the database:

```
$ oc new-app postgresql-ephemeral
```

2. Add the .NET Core application:

```
$ oc new-app dotnet:3.1~https://github.com/redhat-developer/s2i-dotnetcore-persistent-ex#dotnetcore-3.1 --context-dir app
```

3. Add environment variables from the **postgresql** secret and database service name environment variable:

```
$ oc set env dc/s2i-dotnetcore-persistent-ex --from=secret/postgresql -e database-service=postgresql
```

4. Make the application accessible externally:

```
$ oc expose service s2i-dotnetcore-persistent-ex
```

5. Obtain the sharable URL:

```
$ oc get route s2i-dotnetcore-persistent-ex
```

Additional resources

- [s2i-dotnetcore-ex](#) application repository on GitHub

CHAPTER 7. MIGRATION FROM PREVIOUS VERSIONS OF .NET

7.1. MIGRATION FROM PREVIOUS VERSIONS OF .NET

Microsoft provides instructions for migrating from most previous versions of .NET Core.



NOTE

When migrating from ASP.NET Core 2.0 to .NET Core 2.1, the following property should no longer be specified.

<PublishWithAspNetCoreTargetManifest>false</PublishWithAspNetCoreTargetManifest>

It should remain the default value for .NET Core 2.1. Make sure to remove this property from the project file and command line, if it is being specified there.

If you are using a version of .NET that is no longer supported or want to migrate to a newer .NET version to expand functionality, see the following articles:

- [Migrate from ASP.NET Core 3.1 to 5.0](#)
- [Migrate from ASP.NET Core 3.0 to 3.1](#)
- [Migrate from ASP.NET Core 2.2 to 3.0](#)
- [Migrate from ASP.NET Core 2.1 to 2.2](#)
- [Migrate from .NET Core 2.0 to 2.1](#)
- [Migrate from ASP.NET to ASP.NET Core](#)
- [Migrating .NET Core projects from project.json](#)
- [Migrate from project.json to .csproj format](#)



NOTE

If migrating from .NET Core 1.x to 2.0, see the first few related sections in [Migrate from ASP.NET Core 1.x to 2.0](#). These sections provide guidance that is appropriate for a .NET Core 1.x to 2.0 migration path.

7.2. PORTING FROM .NET FRAMEWORK

Refer to the following Microsoft articles when migrating from .NET Framework:

- For general guidelines, see [Porting to .NET Core from .NET Framework](#).
- For porting libraries, see [Porting to .NET Core - Libraries](#).
- For migrating to ASP.NET Core, see [Migrating to ASP.NET Core](#).

Several technologies and APIs present in the .NET Framework are not available in .NET Core and .NET. If your application or library requires these APIs, consider finding alternatives or continue using the .NET Framework. .NET Core and .NET do not support the following technologies and APIs:

- Desktop applications, for example, Windows Forms and Windows Presentation Foundation (WPF)
- Windows Communication Foundation (WCF) servers (WCF clients are supported)
- .NET remoting

Additionally, several .NET APIs can only be used in Microsoft Windows environments. The following list shows examples of these Windows-specific APIs:

- **Microsoft.Win32.Registry**
- **System.AppDomains**
- **System.Drawing**
- **System.Security.Principal.Windows**

Consider using the [.NET Portability Analyzer](#) to identify API gaps and potential replacements.

For example, enter the following command to find out how much of the API used by your .NET Framework application is supported by the newer version of .NET Core:

```
$ dotnet /path/to/ApiPort.dll analyze -f . -r html --target '.NET Framework,Version=<dotnet-framework-version>' --target '.NET Core,Version=<dotnet-version>'
```

Replace `<dotnet-framework-version>` with the .NET Framework version you are currently using. For example, `4.6`. Replace `<dotnet-version>` with the version of .NET Core you plan to migrate to. For example, `3.1`.



IMPORTANT

Several APIs that are not supported in the default version of .NET Core may be available from the [Microsoft.Windows.Compatibility](#) NuGet package. Be careful when using this NuGet package. Some of the APIs provided (such as **Microsoft.Win32.Registry**) only work on Windows, making your application incompatible with Red Hat Enterprise Linux.