



.NET Core 3.1

Getting Started Guide

Legal Notice

Copyright © 2019 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

.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 and OpenShift Container Platform via 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 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 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 Red Hat Enterprise Linux 7 and OpenShift Container Platform versions 3.3 and later.

Table of Contents

CHAPTER 1. USING .NET CORE 3.1 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
1.5. RUN APPLICATIONS ON LINUX CONTAINERS	6
CHAPTER 2. USING .NET CORE 3.1 ON RED HAT OPENSIFT CONTAINER PLATFORM	8
2.1. INSTALLING IMAGE STREAMS	8
2.1.1. Install using oc	8
2.1.2. Install using script	8
2.1.3. Linux/macOS	8
2.1.4. Windows	9
2.2. DEPLOYING APPLICATIONS FROM SOURCE	9
2.3. DEPLOYING APPLICATIONS FROM BINARY ARTIFACTS	9
2.4. USING A JENKINS SLAVE	10
2.5. ENVIRONMENT VARIABLES	12
2.6. SAMPLE APPLICATIONS	15
2.6.1. s2i-dotnetcore-ex	15
2.6.2. s2i-dotnetcore-persistent-ex	15
CHAPTER 3. MIGRATING TO .NET CORE 3.1	17
3.1. MIGRATING FROM PREVIOUS VERSIONS OF .NET CORE	17
3.2. MIGRATING FROM .NET FRAMEWORK TO .NET CORE 3.1	17
3.2.1. Migration Considerations	17
3.2.2. .NET Framework Migration Articles	18

CHAPTER 1. USING .NET CORE 3.1 ON RED HAT ENTERPRISE LINUX

This Getting Started Guide describes how to install .NET Core 3.1 on Red Hat Enterprise Linux (RHEL). See [Red Hat Enterprise Linux documentation](#) for more information about RHEL 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](#)

See the [Red Hat Enterprise Linux Installation Guide](#) for details on how to install RHEL.

See [Red Hat Enterprise Linux Product Documentation page](#) for available RHEL versions.

2. Use the following command to register the system.

```
$ sudo subscription-manager register
```

You can also register the system by following the appropriate steps in [Registering and Unregistering a System](#) in the Red Hat Subscription Management document.

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 3.1 and all of its dependencies.

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

2. Enable the **rh-dotnet31** Software Collection environment so you can run **dotnet** commands in the bash shell.

This procedure installs the .NET Core 3.1 runtime with the latest 3.1 SDK. When a newer SDK becomes available, it automatically installs as a package update.

```
$ scl enable rh-dotnet31 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.



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 the following line to your `~/.bashrc` file.

```
source scl_source enable rh-dotnet31
```

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

```
$ dotnet --info
.NET Core SDK (reflecting any global.json):
Version: 3.1.100
Commit: xxxxxxxxxx

Runtime Environment:
OS Name: rhel
OS Version: 7
OS Platform: Linux
RID: rhel.7-x64
Base Path: /opt/rh/rh-dotnet31/root/usr/lib64/dotnet/sdk/3.1.100/

Host (useful for support):
Version: 3.1.0
Commit: xxxxxxxxxx

.NET Core SDKs installed:
3.1.100 [/opt/rh/rh-dotnet31/root/usr/lib64/dotnet/sdk]

.... omitted
```

1.3. CREATE AN APPLICATION

1. Create a new Console application in a directory called **hello-world**.

```
$ dotnet new console -o hello-world
The template "Console Application" was created successfully.

Processing post-creation actions...
Running 'dotnet restore' on hello-world/hello-world.csproj...
Restore completed in 87.21 ms for /home/<USER>/hello-world/hello-world.csproj.

Restore succeeded.
```

2. Run the project.

```
$ cd hello-world
$ dotnet run
Hello World!
```

1.4. PUBLISH APPLICATIONS

The .NET Core 3.1 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 **rh-dotnet31** Software Collection. On the other hand, SCD uses a runtime built by Microsoft. Running applications outside the **rh-dotnet31** Software Collection may cause issues due to the unavailability of native libraries.

1. Use the following command to publish a framework-dependent application.

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

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

```
$ dotnet restore -r rhel.7-x64
$ dotnet publish -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. This command can be added 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/$APP" "$@"
```

- 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>
```

1.5. RUN APPLICATIONS ON LINUX CONTAINERS

This section shows how to use the **dotnet/dotnet-31-runtime-rhel7** image to run a precompiled application inside a Linux container.

- Create a new mvc project in a directory named **mvc_runtime_example**.

```
$ dotnet new mvc -o mvc_runtime_example
$ cd mvc_runtime_example
```

- Publish the project.

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

- 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
```

- 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.

- Run your image.

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

- View the result in a browser: <http://127.0.0.1:8080>.

[Report a bug](#)

CHAPTER 2. USING .NET CORE 3.1 ON RED HAT OPENSIFT CONTAINER PLATFORM

2.1. INSTALLING IMAGE STREAMS

.NET Core image streams are installed using image stream definitions from [s2i-dotnetcore](#) with the OpenShift client **oc**. A script is available to facilitate removing/installing/updating the image streams.

.NET Core image streams can be defined in the global **openshift** namespace, or locally in a project namespace. To update the **openshift** namespace definitions, you need sufficient permissions.

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.

2.1.1. Install using oc

1. If no pull secret is present in the namespace, you must add one by following the instructions in [Red Hat Container Registry Authentication](#).
2. Run the following commands to list the available .NET Core image streams.

```
$ oc describe is dotnet [-n <namespace>]
```

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

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
```

If no image streams for .NET Core are present, you can install them using:

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

2.1.2. Install using script

The script can be used to install/remove/update .NET Core image streams.

2.1.3. Linux/macOS

1. Download the script from <https://raw.githubusercontent.com/redhat-developer/s2i-dotnetcore/master/install-imagestreams.sh>
2. Login to the OpenShift cluster using the **oc login** command.
3. Install/update the imagestreams.

```
./install-imagestreams.sh --os rhel7 [--namespace <namespace>] [--user  
<subscription_user> --password <subscription_password>]
```

The pull secret can be added by providing the **--user** and **--password** arguments. If a pull secret is already present, these arguments are ignored.

You can run **./install-imagestreams.sh --help** for more information on using this script.

2.1.4. Windows

1. Download the script from <https://raw.githubusercontent.com/redhat-developer/s2i-dotnetcore/master/install-imagestreams.ps1>
2. Login to the OpenShift cluster using the **oc login** command.
3. Install/update the imagestreams.

```
./install-imagestreams.sh --OS rhel7 [--Namespace <namespace>] [-User
<subscription_user> -Password <subscription_password>]
```

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

The pull secret can be added by providing the **-User** and **-Password** arguments. If a pull secret is already present, these arguments are ignored.

You can run **Get-Help .\install-imagestreams.ps1** for more information on using this script.

2.2. DEPLOYING APPLICATIONS FROM SOURCE

1. Run the following commands to deploy the ASP.NET Core application, which is in the **app** folder on the **dotnetcore-3.1** branch of the **redhat-developer/s2i-dotnetcore-ex** GitHub repository.

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

2. Use the **oc logs** command to track progress of the build.

```
$ oc logs -f bc/exampleapp
```

3. View the deployed application once the build is finished.

```
$ 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. DEPLOYING APPLICATIONS FROM BINARY ARTIFACTS

The .NET Core S2I builder image can be used to build an application using binary artifacts that you provide.

1. Publish your application as described in [Publish Applications](#). For example, the following commands create a new web application and publish it.

```
$ dotnet new web -o webapp
$ cd webapp
$ dotnet publish -c Release
```

2. Create a new binary build using the **oc new-build** command.

```
$ oc new-build --name=mywebapp dotnet:3.1 --binary=true
```

3. Start a build using the **oc start-build** command, specifying the path to the binary artifacts on your local machine.

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

4. Create a new application using the **oc new-app** command.

```
$ oc new-app mywebapp
```

2.4. USING A 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.

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

```
$ oc project <projectname>
```

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

- First, the sources are checked out.
- Second, the application is published.
- Third, the image is assembled using a binary build. See [Deploying Applications from Binary Artifacts](#) for additional information about binary builds.

Complete the steps below to configure the example 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-3.1") {
  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/netcoreapp3.1/publish --follow'
    }
  }
}
```

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

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

3. Launch the OpenShift console. Go to **Builds > Pipelines**. The **dotnetapp-build** pipeline is available.
4. Click **Start Pipeline**. 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.
5. When the Jenkins pipeline build completes, go to **Builds > Images**. The **dotnetapp** image is built and available.

2.5. 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.

Variable Name	Description	Default
DOTNET_STARTUP_PROJECT	Selects 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

Variable Name	Description	Default
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 do 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.	
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

Variable Name	Description	Default
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/HTTPS proxy used when building and running the application.	
DOTNET_RM_SRC	When set to true , the source code will not be included in the image.	
DOTNET_SSL_DIRS	Used to specify a list of folders/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
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

Variable Name	Description	Default
DOTNET_PACK	When set to true, creates a tar.gz file at /opt/app-root/app.tar.gz that contains the published application.	

2.6. SAMPLE APPLICATIONS

Two sample applications are available for use with the .NET Core s2i builder.

2.6.1. s2i-dotnetcore-ex

s2i-dotnetcore-ex is the default .NET Core MVC template application.

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 **oc** as follows:

```
# 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

# Make the .NET Core application accessible externally and show the url
$ oc expose service s2i-dotnetcore-ex
$ oc get route s2i-dotnetcore-ex
```

For more information about this application, see <https://github.com/redhat-developer/s2i-dotnetcore-ex/tree/dotnetcore-3.1>.

2.6.2. s2i-dotnetcore-persistent-ex

s2i-dotnetcore-persistent-ex is the a simple CRUD .NET Core web application that stores data in a PostgreSQL database.

The application can be created using the OpenShift client **oc** as follows:

```
# Add the database
$ oc new-app postgresql-ephemeral

# 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

# Add envvars from the the postgresql secret, and database service name envvar.
$ oc set env dc/s2i-dotnetcore-persistent-ex --from=secret/postgresql -e database-service=postgresql

# Make the .NET Core application accessible externally and show the url
$ oc expose service s2i-dotnetcore-persistent-ex
$ oc get route s2i-dotnetcore-persistent-ex
```

For more information about this application, see <https://github.com/redhat-developer/s2i-dotnetcore-persistent-ex/tree/dotnetcore-3.1>.

[Report a bug](#)

CHAPTER 3. MIGRATING TO .NET CORE 3.1

This chapter provides migration information for .NET Core 3.1.

3.1. MIGRATING FROM PREVIOUS VERSIONS OF .NET CORE

See the following Microsoft articles to migrate from previous versions of .NET Core to newer versions of .NET Core.

- [Migrate from .NET Core 2.0 to 2.1](#)
- [Migrate from ASP.NET Core 2.2 to 3.0](#)
- [Migrate from ASP.NET Core 2.1 to 2.2](#)
- [Migrate to ASP.NET Core](#)

3.2. MIGRATING FROM .NET FRAMEWORK TO .NET CORE 3.1

Review the following information to migrate from the .NET Framework.

3.2.1. Migration Considerations

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

- Windows Communication Foundation (WCF) servers (WCF clients are supported)
- .NET remoting

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

- Microsoft.Win32.Registry
- System.AppDomains
- 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 4.6 application is supported by .NET Core 2.1.

```
$ dotnet /path/to/ApiPort.dll analyze -f . -r html --target '.NET Framework,Version=4.6' --target '.NET Core,Version=2.1'
```



IMPORTANT

Several APIs that are not supported in the out-of-the-box 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.

3.2.2. .NET Framework Migration Articles

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](#) .

[Report a bug](#)