.NET Core 3.0

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 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 3.0 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 3.0 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
  1.5. RUN APPLICATIONS ON LINUX CONTAINERS ............................................................ 6

CHAPTER 2. USING .NET CORE 3.0 ON RED HAT OPENSSHIFT CONTAINER PLATFORM ................. 8
  2.1. INSTALLING IMAGE STREAMS .......................................................................................... 8
  2.2. DEPLOYING APPLICATIONS FROM SOURCE .................................................................. 9
  2.3. DEPLOYING APPLICATIONS FROM BINARY ARTIFACTS ............................................... 10
  2.4. USING A JENKINS SLAVE ................................................................................................ 10
  2.5. ENVIRONMENT VARIABLES .......................................................................................... 12
  2.6. SAMPLE APPLICATIONS .................................................................................................. 15

CHAPTER 3. MIGRATING TO .NET CORE 3.0 ............................................................................. 16
  3.1. MIGRATING FROM PREVIOUS VERSIONS OF .NET CORE ............................................... 16
  3.2. MIGRATING FROM .NET FRAMEWORK TO .NET CORE 3.0 .......................................... 16
    3.2.1. Migration Considerations ......................................................................................... 16
    3.2.2. .NET Framework Migration Articles ........................................................................ 17

APPENDIX A. REVISION HISTORY ............................................................................................ 18
CHAPTER 1. USING .NET CORE 3.0 ON RED HAT ENTERPRISE LINUX

This Getting Started Guide describes how to install .NET Core 3.0 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.

2. Use the following command to register the system.

   ```bash
   $ 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.

   ```bash
   $ 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.

   ```bash
   $ 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.

   ```bash
   $ 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.

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

7. Install the scl tool.

   ```bash
   $ sudo yum install scl-utils
   ```
1.2. INSTALL .NET CORE

1. Install .NET Core 3.0 and all of its dependencies.

   $ sudo yum install rh-dotnet30 -y

2. Enable the `rh-dotnet30` Software Collection environment so you can run `dotnet` commands in the bash shell.

   This procedure installs the .NET Core 3.0 runtime with the latest 3.0 SDK. When a newer SDK becomes available, it automatically installs as a package update.

   $ scl enable rh-dotnet30 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-dotnet30` because it may affect other programs. For example, `rh-dotnet30` 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.

   ```bash
   source scl_source enable rh-dotnet30
   ```

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

   $ dotnet --info

   .NET Core SDK (reflecting any global.json):
   Version: 3.0.100
   Commit: xxxxxxxxxx

   Runtime Environment:
   OS Name: rhel
   OS Version: 7
   OS Platform: Linux
   RID: rhel.7-x64
   Base Path: /opt/rh/rh-dotnet30/root/usr/lib64/dotnet/sdk/3.0.100/

   Host (useful for support):
   Version: 3.0.0
   Commit: xxxxxxxxxx

   .NET Core SDKs installed:
   3.0.100 [/opt/rh/rh-dotnet30/root/usr/lib64/dotnet/sdk]
   .... omitted
### 1.3. CREATE AN APPLICATION

1. Create a new Console application in a directory called `hello-world`.
   
   ```sh
   $ 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.
   
   ```sh
   $ cd hello-world
   $ dotnet run
   Hello World!
   ```

### 1.4. PUBLISH APPLICATIONS

The .NET Core 3.0 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 **rh-dotnet30** Software Collection. On the other hand, SCD uses a runtime built by Microsoft. Running applications outside the **rh-dotnet30** Software Collection may cause issues due to the unavailability of native libraries.

1. Use the following command to publish a framework-dependent application.
   
   ```sh
   $ dotnet publish -f netcoreapp3.0 -c Release
   ```

2. Optional: If the application is only for RHEL, trim out the dependencies needed for other platforms with these commands.
   
   ```sh
   $ dotnet restore -r rhel.7-x64
   $ dotnet publish -f netcoreapp3.0 -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.
   
   ```sh
   $ scl enable rh-dotnet30 -- 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.

   ```bash
   #!/bin/bash
   APP=<app>
   SCL=rh-dotnet30
   ```
DIR="$(dirname "$(readlink -f "$0")")"

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

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

```xml
<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-30-runtime-rhel7 image to run a precompiled application inside a Linux container.

1. Create a new mvc project in a directory named mvc_runtime_example.

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

2. Publish the project.

   ```bash
   $ dotnet publish -f netcoreapp3.0 -c Release
   ```

3. Create the Dockerfile.

   ```bash
   $ cat > Dockerfile <<EOF
   FROM registry.redhat.io/dotnet/dotnet-30-runtime-rhel7
   ADD bin/Release/netcoreapp3.0/publish/ .
   CMD ["dotnet", "mvc_runtime_example.dll"]
   EOF
   ```

4. Build your image.

   ```bash
   $ podman build -t dotnet-30-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.

5. Run your image.

   ```bash
   $ podman run -d -p8080:8080 dotnet-30-runtime-example
   ```

6. View the result in a browser: [http://127.0.0.1:8080](http://127.0.0.1:8080).
Report a bug
CHAPTER 2. USING .NET CORE 3.0 ON RED HAT OPENSIFT CONTAINER PLATFORM

2.1. INSTALLING 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 commands to list all available .NET Core image versions.

   `$ oc describe is dotnet -n openshift`
   `$ 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.

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

   `$ oc get secret | grep kubernetes.io/dockerc`

5. To check the contents of a secret, you can decode the .dockercfg or .dockerconfigjson 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:
   eyJyZWdpc3RyeS5yZWRoYXQuaW8iOiJidWxpY2F0aW9uOlNzcGxwdWdpbiJ9.fZ9Mj7s559zjZStvZf22yK
   0=

6. 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`
You need to add a secret if there is no secret listed with credentials for the `registry.redhat.io` server.

7. 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.

```
$ podman 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
```

8. 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. 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-3.0` branch of the `redhat-developer/s2i-dotnetcore-ex` GitHub repository.

```
$ oc new-app --name=exampleapp 'dotnet:3.0~https://github.com/redhat-developer/s2i-dotnetcore-ex#dotnetcore-3.0' --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.
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 logs -f dc/exampleapp

4.

$ 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:3.0 --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/netcoreapp3.0/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 3.0 slave image (`dotnet-30`). 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-30
labels:
```
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-30 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.

```yaml
kind: BuildConfig
apiVersion: v1
metadata:
  name: dotnetapp-build
spec:
  strategy:
    type: JenkinsPipeline
    jenkinsPipelineStrategy:
      jenkinsfile: |
        node("dotnet-30") {
          stage('clone sources') {
            sh "git clone https://github.com/redhat-developer/s2i-dotnetcore-ex --branch dotnetcore-3.0 ."
          }
          stage('publish') {
            dir('app') {
              sh "dotnet publish -c Release"
            }
          }
          stage('create image') {
            dir('app') {
              sh 'oc new-build --name=dotnetapp dotnet:3.0 --binary=true || true'
              sh 'oc start-build dotnetapp --from-dir=bin/Release/netcoreapp3.0/publish --follow'
            }
          }
        }
```

2. Import the `BuildConfig` file to OpenShift.

   ```bash
   $ 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.
<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOTNET_STARTUP_PROJECT</td>
<td>Selects project to run. This must be a project file (for example, <code>csproj</code> or <code>fsproj</code>) or a folder containing a single project file.</td>
<td>.</td>
</tr>
<tr>
<td>DOTNET_ASSEMBLY_NAME</td>
<td>Selects the assembly to run. This must not include the .dll extension. Set this to the output assembly name specified in <code>csproj</code> (PropertyGroup/AssemblyName).</td>
<td>The name of the <code>csproj</code> file</td>
</tr>
<tr>
<td>DOTNET_PUBLISH_READYTORUN</td>
<td>When set to <code>true</code>, 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.</td>
<td>false</td>
</tr>
<tr>
<td>DOTNET_RESTORE_SOURCES</td>
<td>Specifies the space-separated list of NuGet package sources used during the restore operation. This overrides all of the sources specified in the <code>NuGet.config</code> file. This variable cannot be combined with <code>DOTNET_RESTORE_CONFIGFILE</code>.</td>
<td></td>
</tr>
<tr>
<td>DOTNET_RESTORE_CONFIGFILE</td>
<td>Specifies a <code>NuGet.Config</code> file to be used for restore operations. This variable cannot be combined with <code>DOTNET_RESTORE_SOURCES</code>.</td>
<td></td>
</tr>
<tr>
<td>DOTNET_TOOLS</td>
<td>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 <code>@&lt;version&gt;</code>.</td>
<td></td>
</tr>
<tr>
<td>DOTNET_NPM_TOOLS</td>
<td>Specifies a list of NPM packages to install before building the application.</td>
<td></td>
</tr>
<tr>
<td>DOTNET_TEST_PROJECTS</td>
<td>Specifies 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>Variable Name</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>----------</td>
</tr>
<tr>
<td>DOTNET_CONFIGURATION</td>
<td>Runs the application in Debug or Release mode. This value should be either Release or Debug.</td>
<td>Release</td>
</tr>
<tr>
<td>DOTNET_VERBOSITY</td>
<td>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 (quiet, minimal, normal, detailed, and diagnostic).</td>
<td></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>DOTNET_RM_SRC</td>
<td>When set to true, the source code will not be included in the image.</td>
<td></td>
</tr>
<tr>
<td>DOTNET_SSL_DIRS</td>
<td>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).</td>
<td></td>
</tr>
<tr>
<td>NPM_MIRROR</td>
<td>Uses a custom NPM registry mirror to download packages during the build process.</td>
<td></td>
</tr>
<tr>
<td>ASPNETCORE_URLS</td>
<td>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.</td>
<td>http://*:8080</td>
</tr>
<tr>
<td>DOTNET_RESTORE_DISABLE_PARALLEL</td>
<td>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.</td>
<td>false</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_INCREMENTAL</td>
<td>When set to true, the NuGet packages will be kept so they can be re-used for an incremental build.</td>
<td>false</td>
</tr>
<tr>
<td>DOTNET_PACK</td>
<td>When set to true, creates a tar.gz file at /opt/app-root/app.tar.gz that contains the published application.</td>
<td></td>
</tr>
</tbody>
</table>

## 2.6. 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. The application is built in `dotnet/dotnet-30-rhel7`. The result is deployed in `dotnet/dotnet-30-runtime-rhel7`. Note that chained builds are not supported on OpenShift Online.
- **dotnet-pgsql-persistent**: This is the Microsoft ASP.NET Core MusicStore sample application using a PostgreSQL backend.

**NOTE**

This application uses .NET Core 2.1 LTS.

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.

```
$ 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
```

Report a bug
CHAPTER 3. MIGRATING TO .NET CORE 3.0

This chapter provides migration information for .NET Core 3.0.

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 3.0

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

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
## 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>06/14/2018</td>
<td>2.1</td>
<td>Les Williams</td>
<td>Generally available</td>
</tr>
<tr>
<td>08/01/2018</td>
<td>2.1</td>
<td>Toby Drake</td>
<td>Added Chapter 3 to provide migration instructions</td>
</tr>
<tr>
<td>08/24/2018</td>
<td>2.1</td>
<td>Toby Drake</td>
<td>Added steps to enable a user to get new image streams</td>
</tr>
<tr>
<td>09/18/2018</td>
<td>2.1</td>
<td>Toby Drake</td>
<td>Revised Section 2.1 to include <code>--n openshift</code> in a command for listing .NET Core image versions. Modified the <code>grep</code> command to enable better search results.</td>
</tr>
<tr>
<td>Date</td>
<td>Version</td>
<td>Author</td>
<td>Changes</td>
</tr>
<tr>
<td>------------</td>
<td>---------</td>
<td>----------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>10/12/2018</td>
<td>2.1</td>
<td>Toby Drake</td>
<td>Added DOTNET_SSL_DIRS and DOTNET_RM_SRC to Environment Variables. Added Deploy Applications from Binary Artifacts.</td>
</tr>
<tr>
<td>11/08/2018</td>
<td>2.1</td>
<td>Toby Drake</td>
<td>Changed references from docker to podman. Changed registry server to registry.redhat.io. Added procedure to set up Jenkins master-slave pipeline. See Using a Jenkins Slave.</td>
</tr>
<tr>
<td>12/04/2018</td>
<td>2.2</td>
<td>Toby Drake</td>
<td>Generally available</td>
</tr>
<tr>
<td>12/06/2018</td>
<td>2.2</td>
<td>Les Williams</td>
<td>Added link for migrating from ASP.NET Core 2.1 to 2.2</td>
</tr>
<tr>
<td>04/16/2019</td>
<td>2.2</td>
<td>Les Williams</td>
<td>Revised environment variables section for DOTNET_INCREMENTAL and DOTNET PACK variables</td>
</tr>
</tbody>
</table>

Report a bug