.NET 8.0

Getting started with .NET on RHEL 9

Installing and running .NET 8.0 on RHEL 9

Last Updated: 2023-11-30

.NET 8.0 Getting started with .NET on RHEL 9

Installing and running .NET 8.0 on RHEL 9

Legal Notice

Abstract

This guide describes how to install and run .NET 8.0 on RHEL 9.

Table of Contents

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 feedback on our documentation. Let us know how we can improve it.

Submitting feedback through Jira (account required)

1. Log in to the Jira website.
2. Click **Create** in the top navigation bar
3. Enter a descriptive title in the **Summary** field.
4. Enter your suggestion for improvement in the **Description** field. Include links to the relevant parts of the documentation.
5. Click **Create** at the bottom of the dialogue.

CHAPTER 1. INTRODUCING .NET 8.0

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

.NET 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 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 8.0 is supported on RHEL 8.9 and later, RHEL 9.3 and later, and supported OpenShift Container Platform versions.

CHAPTER 2. INSTALLING .NET 8.0

.NET 8.0 is included in the AppStream repositories for RHEL 9. The AppStream repositories are enabled by default on RHEL 9 systems.

You can install the .NET 8.0 runtime with the latest 8.0 Software Development Kit (SDK). When a newer SDK becomes available for .NET 8.0, you can install it by running **sudo yum install**.

Prerequisites

- Installed and registered RHEL 9.3 with attached subscriptions.
  For more information, see Performing a standard RHEL 9 installation .

Procedure

- Install .NET 8.0 and all of its dependencies:

```
$ sudo yum install dotnet-sdk-8.0 -y
```

Verification steps

- Verify the installation:

```
$ dotnet --info
```

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

CHAPTER 3. CREATING AN APPLICATION USING .NET 8.0

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:

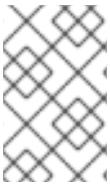
```
Hello World!
```

CHAPTER 4. PUBLISHING APPLICATIONS WITH .NET 8.0

.NET 8.0 applications can be published to use a shared system-wide version of .NET or to include .NET.

The following methods exist for publishing .NET 8.0 applications:

- Self-contained deployment (SCD) - The application includes .NET. This method uses a runtime built by Microsoft.
- 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, built by Red Hat, that uses a set of native dependencies.

Prerequisites

- Existing .NET application.
  For more information about how to create a .NET application, see Creating an application using .NET.

4.1. PUBLISHING .NET APPLICATIONS

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

Procedure

1. Publish the framework-dependent application:

   ```
   $ dotnet publish my-app -f net8.0
   ```

   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 publish my-app -f net8.0 -r rhel.9-architecture --self-contained false
   ```

   - Replace *architecture* based on the platform you are using:
     - For Intel: **x64**
     - For IBM Z and LinuxONE: **s390x**
     - For 64-bit Arm: **arm64**
     - For IBM Power: **ppc64le**

CHAPTER 5. RUNNING .NET 8.0 APPLICATIONS IN CONTAINERS

Use the **ubi8/dotnet-80-runtime** image to run a .NET application inside a Linux container.

The following example uses podman.

Procedure

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

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

2. Publish the project:

```
$ dotnet publish mvc_runtime_example -f net8.0 /p:PublishProfile=DefaultContainer
/p:ContainerBaseImage=registry.access.redhat.com/ubi8/dotnet-80-runtime:latest
```

3. Run your image:

```
$ podman run --rm -p8080:8080 mvc_runtime_example
```

Verification steps

- View the application running in the container:

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

CHAPTER 6. USING .NET 8.0 ON OPENSHIFT CONTAINER PLATFORM

6.1. OVERVIEW

NET images are added to OpenShift by importing imagestream definitions from s2i-dotnetcore.

The imagestream definitions include the **dotnet** imagestream which contains sdk images for different supported versions of .NET. Life Cycle and Support Policies for the .NET Program provides an up-to-date overview of supported versions.

Version	Tag	Alias
.NET 6.0	dotnet:6.0-ubi8	dotnet:6.0
.NET 7.0	dotnet:7.0-ubi8	dotnet:7.0
.NET 8.0	dotnet:8.0-ubi8	dotnet:8.0

The sdk images have corresponding runtime images which are defined under the **dotnet-runtime** imagestream.

The container images work across different versions of Red Hat Enterprise Linux and OpenShift. The UBI-8 based images (suffix -ubi8) are hosted on the **registry.access.redhat.com** and do not require authentication.

6.2. INSTALLING .NET IMAGE STREAMS

To install .NET 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.

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

Procedure

1. Install (or update) the image streams:

```
$ oc apply [-n namespace] -f
https://raw.githubusercontent.com/redhat-developer/s2i-
dotnetcore/main/dotnet_imagestreams.json
```

6.3. 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 **dotnet-8.0** 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:8.0-ubi8~https://github.com/redhat-developer/s2i-dotnetcore-ex#dotnet-8.0' --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
```

6.4. DEPLOYING APPLICATIONS FROM BINARY ARTIFACTS USINGoc

You can use .NET 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:8.0-ubi8 --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/net8.0/publish
```

3. Create a new application:

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

6.5. ENVIRONMENT VARIABLES FOR .NET 8.0

The .NET images support several environment variables to control the build behavior of your .NET 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_READYTORUN	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	**Deprecated**: Use **SSL_CERT_DIR** instead	
SSL_CERT_DIR	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.

This application is used as the example application by the .NET 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 application:

```
$ oc new-app dotnet:8.0-ubi8~https://github.com/redhat-developer/s2i-dotnetcore-ex#dotnet-8.0 --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 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 application:

```
$ oc new-app dotnet:8.0-ubi8~https://github.com/redhat-developer/s2i-dotnetcore-persistent-ex#dotnet-8.0 --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