Red Hat Enterprise Linux 8

RHEL 8 での.NET アプリケーションの開発

Red Hat Enterprise Linux 8 で .NET アプリケーションを開発する .NET Core 3.1 のインストールおよび実行

Last Updated: 2020-03-19

Red Hat Enterprise Linux 8 RHEL 8 での.NET アプリケーションの開発

Red Hat Enterprise Linux 8 で .NET アプリケーションを開発する .NET Core 3.1 のインストールおよび実行

法律上の通知

概要

.NET Core は、自動メモリー管理と最新のプログラミング言語を備えた汎用開発プラットフォームです。これにより、ユーザーは高品質のアプリケーションを効率的に構築できます。.NET Core は、認定済みのコンテナーを介して Red Hat Enterprise Linux および OpenShift Container Platform で利用できます。 .NET Core には次の機能があります。 マイクロサービスベースのアプローチに従う機能。一部のコンポーネントは .NET で構築され、他のコンポーネントは Java で構築されますが、すべてが Red Hat Enterprise Linux および OpenShift Container Platform でサポートされている共通プラットフォームで実行できます。 Microsoft Windows で新しい .NET Core ワークロードをより簡単に開発する能力。Red Hat Enterprise Linux または Windows Server のいずれかにデプロイして実行できます。 異機種環境のデータセンター。基盤となるインフラストラクチャーが

Windows Server にのみ依存することなく .NET アプリケーションを実行できます。 .NET Core 3.1 は、Red Hat Enterprise Linux 7、Red Hat Enterprise Linux 8、および OpenShift Container Platform バージョン 3.3 以降でサポートされています。

目次

はじめに 3

第1章 RED HAT ENTERPRISE LINUX での .NET CORE 3.1 の使用 4
1.1. .NET CORE のインストール 4
1.2. アプリケーションの作成 4
1.3. アプリケーションの公開 4
1.4. コンテナーでアプリケーションの実行 5

第2章 RED HAT OPENSHIFT CONTAINER PLATFORM での .NET CORE 3.1 の使用 6
2.1. イメージストリームのインストール 6
2.2. アプリケーションのデプロイメント 7
2.3. 環境変数 8
2.4. サンプルアプリケーション 11

第3章 .NET CORE 3.1 への移行 13
3.1. .NET CORE の以前のバージョンからの移行 13
3.2. .NET FRAMEWORK から .NET CORE 3.1 への移行 13

はじめに

このガイドでは、.NET Core 3.1 を Red Hat Enterprise Linux 8 (RHEL) にインストールする方法を説明します。RHEL 8 の詳細は、Red Hat Enterprise Linux のドキュメント を参照してください。

第1章 RED HAT ENTERPRISE LINUX での .NET CORE 3.1 の使用

1.1. .NET CORE のインストール

この手順では、最新の 3.1 SDK を使用して .NET Core 3.1 ランタイムをインストールします。新しい SDK が使用可能になると、パッケージの更新として自動的にインストールされます。

手順

.NET Core 3.1 は、RHEL 8 の AppStream リポジトリーに含まれています。AppStream リポジトリーは、RHEL 8 システムでデフォルトで有効になっています。

1. .NET Core 3.1 とそのすべての依存関係をインストールします。

```
$ sudo yum install dotnet-sdk-3.1 -y
```

2. 次のコマンドを実行して、インストールを確認します。

```
$ dotnet --info
```

1.2. アプリケーションの作成

手順

1. **hello-world** という名前のディレクトリーに、新しいコンソールアプリケーションを作成します。

```
$ 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. プロジェクトを実行します。

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

1.3. アプリケーションの公開

.NET Core 3.1 アプリケーションを公開して、共有されたシステム全体で使用される .NET Core を使用するか、.NET Core を追加できます。この 2 種類のデプロイメントは、それぞれフレームワークに依存するデプロイメント (FDD) および自己完結型デプロイメント (SCD) と呼ばれます。

Red Hat Enterprise Linux では、FDD による公開が推奨されます。この方法により、アプリケーションが Red Hat により構築された最新バージョンの .NET Core を使用し、特定セットのネイティブ依存関係が含まれるようになります。一方、SCD は Microsoft が作成したランタイムを使用します。

手順

1. フレームワーク依存アプリケーションを公開するには、次のコマンドを使用します。

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

2. **任意:** アプリケーションが RHEL 専用の場合は、次のコマンドを使用してその他のプラットフォームに必要な依存関係を削除します。

```
$ dotnet restore -r rhel.8-x64
$ dotnet publish -f netcoreapp3.1 -c Release -r rhel.8-x64 --self-contained false
```

1.4. コンテナーでアプリケーションの実行

本セクションでは、**ubi8/dotnet-31-runtime** イメージを使用して、コンテナー内でプリコンパイルされたアプリケーションを実行する方法を示します。

手順

1. **mvc_runtime_example** という名前のディレクトリーに新しい MVC プロジェクトを作成します。

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

2. プロジェクトを公開します。

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

3. **Dockerfile** を作成します。

```
$ cat > Dockerfile <<EOF
FROM registry.access.redhat.com/ubi8/dotnet-31-runtime

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

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

4. イメージを構築します。

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

5. イメージを実行します。

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

6. ブラウザー (**http://127.0.0.1:8080**) で結果を表示します。

第2章 RED HAT OPENSHIFT CONTAINER PLATFORM での .NET CORE 3.1 の使用

2.1. イメージストリームのインストール

.NET Core イメージストリームは、s2i-dotnetcore のイメージストリーム定義と OpenShift クライアントバイナリー (**oc**) を使用してインストールされます。イメージストリームの削除、インストール、および更新を容易にするスクリプトが利用できます。

.NET Core イメージストリームは、グローバルな **openshift** 名前空間で定義するか、プロジェクト名前空間でローカルに定義できます。**openshift** 名前空間の定義を更新するには、十分な権限が必要です。

2.1.1. oc を使用したイメージストリームのインストール

前提条件

- 名前空間に存在する既存のプルシークレット名前空間にプルシークレットが存在しない場合は、「Red Hat コンテナーレジストリーの認証」の手順に従ってプルシークレットを追加する必要があります。

手順

1. 利用可能な .NET Core イメージストリームの一覧を表示します。

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

   出力には、インストールされているイメージが表示されます。イメージがインストールされていない場合は、**Error from server (NotFound)** メッセージが表示されます。

2. .NET Core イメージストリームをインストールします。

```
$ oc create -f https://raw.githubusercontent.com/redhat-developer/s2i-dotnetcore/master/dotnet_imagestreams_rhel8.json
```

3. 既存の .NET Core イメージストリームの新しいバージョンを含めます。

```
$ oc replace -f https://raw.githubusercontent.com/redhat-developer/s2i-dotnetcore/master/dotnet_imagestreams_rhel8.json
```

2.1.2. スクリプトを使用したイメージストリームのインストール

このスクリプトを使用して、.NET Core イメージストリームをインストール、削除、および更新できます。

2.1.2.1. Linux/macOS

手順

1. スクリプトをダウンロードします。

2. **oc login** コマンドを使用して、OpenShift クラスターにログインします。

3. イメージストリームをインストールまたは更新します。

```
./install-imagestreams.sh --os rhel8 [--namespace <namespace>]  [--user
<subscription_user> --password <subscription_password>]
```

プルシークレットを追加するには、**--user** 引数および **--password** 引数を指定します。プルシークレットが存在する場合は、指定した引数が無視されます。

このスクリプトの使用の詳細は、**./install-imagestreams.sh --help** を実行してください。

2.1.2.2. Windows

手順

1. スクリプトをダウンロードします。
2. **oc login** コマンドを使用して、OpenShift クラスターにログインします。
3. イメージストリームをインストールまたは更新します。

```
./install-imagestreams.sh --OS rhel8 [--Namespace <namespace>]  [-User
<subscription_user> -Password <subscription_password>]
```

注記

PowerShell の **ExecutionPolicy** では、このスクリプトの実行が禁止される場合があります。ポリシーを緩和するには、**Set-ExecutionPolicy -Scope Process -ExecutionPolicy Bypass -Force** を実行します。

プルシークレットを追加するには、**-User** 引数および **-Password** 引数を指定します。プルシークレットが存在する場合は、指定した引数が無視されます。

このスクリプトの使用の詳細は、**Get-Help .\install-imagestreams.ps1** を実行してください。

2.2. アプリケーションのデプロイメント

2.2.1. ソースからアプリケーションのデプロイメント

手順

1. 以下のコマンドを実行して、GitHub リポジトリー **redhat-developer/s2i-dotnetcore-ex** の **dotnetcore-3.1** ブランチの **app** ディレクトリーに、ASP.NET Core アプリケーションをデプロイします。

```
$ 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. **oc logs** コマンドを使用して、ビルドの進行状況を追跡します。

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

3. ビルドが完了したら、デプロイされたアプリケーションを表示します。

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

4. これで、プロジェクト内でアプリケーションにアクセスできます。プロジェクトを外部からアクセス可能にするには、**oc expose** コマンドを使用します。次に、**oc get routes** を使用して URL を検索できます。

```
$ oc expose svc/exampleapp
$ oc get routes
```

2.2.2. バイナリーアーティファクトからアプリケーションのデプロイ

.NET Core Source-to-Image (S2I) ビルダーイメージを使用して、提供するバイナリーアーティファクトを使用してアプリケーションをビルドできます。

手順

1. 「アプリケーションの公開」の説明に従ってアプリケーションを公開します。たとえば、次のコマンドは新しい Web アプリケーションを作成して公開します。

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

2. **oc new-build** コマンドを使用して、新しいバイナリービルドを作成します。

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

3. **oc start-build** コマンドを使用してビルドを開始し、ローカルマシンのバイナリーアーティファクトへのパスを指定します。

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

4. **oc new-app** コマンドを使用して新しいアプリケーションを作成します。

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

2.3. 環境変数

.NET Core イメージは、.NET Core アプリケーションのビルド動作を制御する多数の環境変数に対応しています。これらの変数は、ビルド構成の一部として設定することも、アプリケーションソースコードリポジトリーの **.s2i/environment** ファイルに追加することもできます。

変数名	説明	デフォルト
DOTNET_STARTUP_PROJECT	実行するプロジェクトを選択します。これは、プロジェクトファイル (**csproj**、**fsproj** など) またはプロジェクトファイルを 1 つ含むディレクトリーである必要があります。	.

変数名	説明	デフォルト
DOTNET_ASSEMBLY_NAME	実行するアセンブリーを選択します。これには **.dll** 拡張子を含めないでください。これを、**csproj** で指定した出力アセンブリー名 (PropertyGroup/AssemblyName) に設定します。	**csproj** ファイルの名前
DOTNET_PUBLISH_READRYTORUN	**true** に設定すると、アプリケーションは事前にコンパイルされます。これにより、アプリケーションの読み込み時に JIT が必要な作業量が削減されるため、起動時間が短縮されます。	**false**
DOTNET_RESTORE_SOURCES	復元操作中に使用される NuGet パッケージソースのスペース区切り一覧を指定します。これにより、**NuGet.config** ファイルで指定されたすべてのソースが上書きされます。この変数を **DOTNET_RESTORE_CONFIGFILE** と組み合わせることはできません。	
DOTNET_RESTORE_CONFIGFILE	復元操作に使用される **NuGet.Config** ファイルを指定します。この変数を **DOTNET_RESTORE_SOURCES** と組み合わせることはできません。	
DOTNET_TOOLS	アプリをビルドする前にインストールする .NET ツールの一覧を指定します。**@<version>** でパッケージ名を保留することにより、特定のバージョンをインストールできます。	
DOTNET_NPM_TOOLS	アプリケーションをビルドする前にインストールする NPM パッケージの一覧を指定します。	
DOTNET_TEST_PROJECTS	テストするテストプロジェクトの一覧を指定します。これは、複数のプロジェクトファイル、またはプロジェクトファイルを 1 つ含む複数のディレクトリーである必要があります。アイテムごとに **dotnet test** が呼び出されます。	

変数名	説明	デフォルト
DOTNET_CONFIGURATION	Debug モードまたは Release モードでアプリケーションを実行します。この値は、**Release** または **Debug** でなければなりません。	**Release**
DOTNET_VERBOSITY	**dotnet build** コマンドの詳細度を指定します。設定すると、環境変数はビルドの開始時に出力されます。この変数は、msbuild の詳細度 (**q[uiet]**、**m[inimal]**、**n[ormal]**、**d[etailed]**、および **diag[nostic]**) のいずれかに設定できます。	
HTTP_PROXY, HTTPS_PROXY	アプリケーションをビルドおよび実行するときにそれぞれ使用される HTTP または HTTPS プロキシーを構成します。	
DOTNET_RM_SRC	**true** に設定すると、ソースコードはイメージに含まれません。	
DOTNET_SSL_DIRS	信頼する追加の SSL 証明書を含むディレクトリーまたはファイルの一覧を指定するために使用されます。証明書は、ビルド中に実行する各プロセスと、ビルド後のイメージで実行するすべてのプロセス (ビルドされたアプリケーションを含む) により信頼されます。項目は、絶対パス (/ で始まる) またはソースリポジトリーのパス (証明書など) にすることができます。	
NPM_MIRROR	ビルドプロセス中にカスタム NPM レジストリミラーを使用してパッケージをダウンロードします。	
ASPNETCORE_URLS	この変数は **http://*:8080** に設定され、イメージにより公開されるポートを使用するように ASP.NET Core を設定します。これを変更することは推奨されません。	**http://*:8080**
DOTNET_RESTORE_DISABLE_PARALLEL	**true** に設定すると、複数のプロジェクトを並行して復元できなくなります。これにより、ビルドコンテナーが低い CPU 制限で実行している場合の復元タイムアウトエラーが減少します。	**false**

変数名	説明	デフォルト
DOTNET_INCREMENTAL	**true** に設定すると、NuGet パッケージは保持され、インクリメンタルビルドに再利用できます。	**false**
DOTNET_PACK	**true** に設定すると、公開アプリケーションを含む **tar.gz** ファイルが **/opt/app-root/app.tar.gz** に作成されます。	

2.4. サンプルアプリケーション

.NET Core S2I ビルダーでは、2 つのサンプルアプリケーションを使用できます。

2.4.1. MVC サンプルアプリケーションの作成

s2i-dotnetcore-ex は、デフォルトの.NET Core Model、View、Controller (MVC) テンプレートアプリケーションです。

このアプリケーションは、.NET Core S2I イメージによってサンプルアプリケーションとして使用され、**Try Example** リンクを使用して OpenShift UI から直接作成できます。

アプリケーションは、OpenShift クライアントバイナリー (**oc**) を使用して作成することもできます。

手順

oc を使用してサンプルアプリケーションを作成するには、以下を行います。

1. .NET Core アプリケーションを追加します。

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

2. .NET Core アプリケーションが外部からアクセスできるようにし、URL を表示します。

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

関連情報

- このアプリケーションの詳細は、GitHub の **s2i-dotnetcore-ex リポジトリー** を参照してください。

2.4.2. CRUD サンプルアプリケーションの作成

s2i-dotnetcore-persistent-ex は、PostgreSQL データベースにデータを格納する単純な Create、Read、Update、Delete (CRUD) の .NET Core Web アプリケーションです。

手順

アプリケーションは、次のように OpenShift クライアント **oc** を使用して作成できます。

1. データベースを追加します。

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

2. .NET Core アプリケーションを追加します。

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

3. **postgresql** シークレットおよびデータベースサービス名環境変数から環境変数を追加します。

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

4. .NET Core アプリケーションが外部からアクセスできるようにし、URL を表示します。

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

関連情報

- このアプリケーションの詳細は、GitHub の **s2i-dotnetcore-persistent-ex** リポジトリー を参照してください。

第3章 .NET CORE 3.1 への移行

この章では、.NET Core 3.1 の移行情報を提供します。

3.1. .NET CORE の以前のバージョンからの移行

次の Microsoft の記事を参照して、以前のバージョンの .NET Core から新しいバージョンの .NET Core に移行してください。

- .NET Core 2.0 から 2.1 への移行
- ASP.NET Core 2.2 から 3.0 への移行
- ASP.NET Core 2.1 から 2.2 への移行
- への移行

3.2. .NET FRAMEWORK から .NET CORE 3.1 への移行

.NET Framework から移行するには、次の情報を確認してください。

3.2.1. 移行に関する考慮事項

.NET Framework に存在するいくつかの技術と API は、.NET Core では使用できません。アプリケーションまたはライブラリーにこれらの API が必要な場合は、代替を見つけることを検討するか、.NET Framework の使用を継続してください。.NET Core は、次の技術と API をサポートしていません。

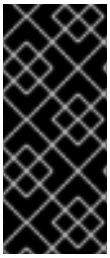
- Windows Communication Foundation (WCF) サーバー (WCF クライアントがサポートされています)
- .NET リモート処理

さらに、多くの .NET API は、Microsoft Windows 環境でのみ使用できます。次の一覧に、この Windows 固有の API の例をいくつか示します。

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

.NET Portability Analyzer を使用して、API の相違点と交換の可能性を特定することを検討してください。たとえば、次のコマンドを入力して、.NET Framework 4.6 アプリケーションで使用される API が .NET Core 2.1 でどの程度サポートされているかを確認します。

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



重要

.NET Core のデフォルトバージョンでサポートされないいくつかの API は、Microsoft.Windows.Compatibility NuGet パッケージで利用できます。この NuGet パッケージを使用するときは注意してください。提供されている API の一部 (**Microsoft.Win32.Registry** など) は Windows でのみ動作し、アプリケーションが Red Hat Enterprise Linux と互換性がないようにします。

3.2.2. .NET Framework の移行に関する記事

.NET Framework から移行する場合は、次の Microsoft の記事を参照してください。

- 一般的なガイドラインは、「Porting to .NET Core from .NET Framework」 を参照してください。
- ライブラリの移植は、「Porting to .NET Core - Libraries」 を参照してください。
- ASP.NET Core への移行は、「Migrating to ASP.NET Core」 を参照してください。