.NET 8.0

RHEL 9 で .NET を使い始める

RHEL 9 での .NET 8.0 のインストールおよび実行

Last Updated: 2023-12-29

.NET 8.0 RHEL 9 で .NET を使い始める

RHEL 9 での .NET 8.0 のインストールおよび実行

法律上の通知

概要

本ガイドでは、RHEL 9 に .NET 8.0 をインストールして実行する方法を説明します。

目次

多様性を受け入れるオープンソースの強化

Red Hat では、コード、ドキュメント、Web プロパティーにおける配慮に欠ける用語の置き換えに取り組んでいます。まずは、マスター (master)、スレーブ (slave)、ブラックリスト (blacklist)、ホワイトリスト (whitelist) の 4 つの用語の置き換えから始めます。この取り組みは膨大な作業を要するため、今後の複数のリリースで段階的に用語の置き換えを実施して参ります。詳細は、Red Hat CTO である Chris Wright のメッセージ をご覧ください。

RED HAT ドキュメントへのフィードバック (英語のみ)

Red Hat ドキュメントに関するご意見やご感想をお寄せください。また、改善点があればお知らせください。

Jira からのフィードバック送信 (アカウントが必要)

1. Jira の Web サイトにログインします。
2. 上部のナビゲーションバーで **Create** をクリックします。
3. **Summary** フィールドにわかりやすいタイトルを入力します。
4. **Description** フィールドに、ドキュメントの改善に関するご意見を記入してください。ドキュメントの該当部分へのリンクも追加してください。
5. ダイアログの下部にある **Create** をクリックします。

第1章 .NET 8.0 の概要

.NET 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。.NET を使用することで、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 は、認定済みのコンテナーを介して Red Hat Enterprise Linux (RHEL) および OpenShift Container Platform で利用できます。

.NET 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 RHEL および OpenShift Container Platform でサポートされている共通のプラットフォームで実行できます。
- Microsoft Windows で新しい .NET ワークロードをより簡単に開発する機能。RHEL または Windows Server のいずれかにアプリケーションを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 8.0 は、RHEL 8.9 以降および RHEL 9.3 以降、および対象の OpenShift Container Platform バージョンでサポートされています。

第2章 .NET 8.0 のインストール

.NET 8.0 は、RHEL 9 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 9 システムでデフォルトで有効になっています。

.NET 8.0 ランタイムは、最新の 8.0 Software Development Kit (SDK) でインストールできます。新しい SDK が .NET 8.0 で利用可能になったら、**sudo yum install** を実行してインストールできます。

前提条件

- サブスクリプションを割り当てて、RHEL 9.3 をインストールして登録する。
  詳細は、標準的な RHEL 9 インストールの実行 を参照してください。

手順

- .NET 8.0 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-8.0 -y
```

検証手順

- インストールを確認します。

```
$ dotnet --info
```

  出力は、.NET インストールおよび環境の関連情報を返します。

第3章 .NET 8.0 を使用したアプリケーションの作成

C# **hello-world** アプリケーションを作成する方法を学びます。

手順

1. **my-app** という名前のディレクトリーに、新しい Console アプリケーションを作成します。

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

返される出力は以下のとおりです。

```
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.
```

単純な **Hello World** コンソールアプリケーションが、テンプレートから作成されます。アプリケーションは指定の **my-app** ディレクトリーに保存されます。

検証手順

- プロジェクトを実行します。

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

返される出力は以下のとおりです。

```
Hello World!
```

第4章 .NET 8.0 でのアプリケーションの公開

.NET 8.0 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 を使用するか、.NET を追加できます。

.NET 8.0 アプリケーションを公開するには、以下の方法があります。

- SCD (自己完結型デプロイメント): アプリケーションには .NET が含まれます。この方法では、Microsoft が構築したランタイムを使用します。
- フレームワーク依存デプロイメント (FDD): アプリケーションは、共有されたシステム全体の .NET バージョンを使用します。

注記

RHEL にアプリケーションを公開する場合、Red Hat では FDD を使用することを推奨しています。これは、アプリケーションが、Red Hat が構築した最新バージョンの .NET を使用していることを保証するためです。これは、特定のネイティブ依存関係のセットを使用します。

前提条件

- 既存の .NET アプリケーション。
  .NET アプリケーションの作成方法は、.NET を使用したアプリケーションの作成 を参照してください。

4.1. .NET アプリケーションの公開

以下の手順では、フレームワーク依存アプリケーションを公開する方法を概説します。

手順

1. フレームワーク依存アプリケーションを公開します。

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

   my-app は、公開するアプリケーションの名前に置き換えます。

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

   - **architecture** は、使用しているプラットフォームに基づいて置き換えます。
     - Intel の場合: **x64**
     - IBM Z および LinuxONE の場合: **s390x**
     - 64 ビット Arm の場合: **arm64**
     - IBM Power の場合: **ppc64le**

第5章 コンテナーでの .NET 8.0 アプリケーションの実行

ubi8/dotnet-80-runtime イメージを使用して、.NET コンテナーで事前コンパイルされたアプリケーションを実行します。

以下の例では podman を使用しています。

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 --output mvc_runtime_example
```

2. プロジェクトを公開します。

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

3. イメージを実行します。

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

検証手順

- コンテナーで実行されているアプリケーションを表示します。

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

第6章 OPENSHIFT CONTAINER PLATFORM での .NET 8.0 の使用

6.1. 概要

NET イメージは、s2i-dotnetcore からイメージストリーム定義をインポートすることで OpenShift に追加されます。

イメージストリーム定義には、サポートされる異なるバージョンの .NET の sdk イメージが含まれる **dotnet** イメージストリームが含まれます。.NET プログラムのライフサイクルおよびサポートポリシーでは、サポートされているバージョンの最新の情報をまとめています。

バージョン	タグ	エイリアス
.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

sdk イメージには、**dotnet-runtime** イメージストリームで定義される対応するランタイムイメージがあります。

コンテナーイメージは、Red Hat Enterprise Linux と OpenShift の異なるバージョン間で機能します。UBI-8 ベースのイメージ (suffix -ubi8) は **registry.access.redhat.com** でホストされ、認証は必要ありません。

6.2. .NET イメージストリームのインストール

.NET イメージストリームをインストールするには、s2i-dotnetcore のイメージストリーム定義と OpenShift Client (**oc**) バイナリーを使用してインストールされます。イメージストリームは、Linux、Mac、Windows からインストールできます。

.NET イメージストリームは、グローバルな **openshift** namespace で定義するか、プロジェクト namespace でローカルにストリームします。**openshift** namespace の定義を更新するには、十分な権限が必要です。

手順

1. イメージストリームをインストール (または更新) します。

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

6.3. oc を使用したソースからのアプリケーションのデプロイメント

以下の例では、**oc** を使用した **example-app** アプリケーションのデプロイ方法を説明します。これは、**redhat-developer/s2i-dotnetcore-ex** GitHub リポジトリーの **dotnet-8.0** ブランチの **app** ディレクトリーにあります。

手順

1. 新しい OpenShift プロジェクトを作成します。

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

2. ASP .NET Core アプリケーションを追加します。

```
$ 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. ビルドの進捗を追跡します。

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

4. 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/example-app
```

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。

5. **オプション**: プロジェクトを外部からアクセス可能にします。

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

6. 共有可能な URL を取得します。

```
$ oc get routes
```

6.4. oc を使用した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 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。

前提条件

1. 公開済みアプリケーション。
   詳細は以下を参照してください。

手順

1. 新しいバイナリービルドを作成します。

```
$ oc new-build --name=my-web-app dotnet:8.0-ubi8 --binary=true
```

2. 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 my-web-app --from-dir=bin/Release/net8.0/publish
```

3. 新規アプリケーションを作成します。

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

6.5. .NET 8.0 の環境変数

.NET イメージは、.NET アプリケーションのビルド動作を制御する複数の環境変数をサポートします。これらの変数はビルド設定の一部として設定したり、アプリケーションのソースコードリポジトリーの **.s2i/environment** ファイルに追加できます。

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。	**.**
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_READYTORUN	**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リストを指定します。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。	

変数名	説明	デフォルト
DOTNET_TOOLS	アプリをビルドする前にインストールする .NET ツールのリストを指定します。**@<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 パッケージのリストを指定します。	
DOTNET_TEST_PROJECTS	テストするテストプロジェクトのリストを指定します。これは、プロジェクトファイルまたは、単一のプロジェクトファイルを含むディレクトリーである必要があります。各項目に対して **dotnet test** が呼び出されます。	
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 プロキシーを設定します。	
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	**非推奨**: 代わりに **SSL_CERT_DIR** を使用してください	

変数名	説明	デフォルト
SSL_CERT_DIR	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リストを指定します。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 レジストリー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、CPU 制限の値が低く設定された状態で、ビルドコンテナーが実行されている場合にも復元タイムアウトのエラーが減少します。	**false**
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。	

6.6. MVC サンプルアプリケーションの作成

s2i-dotnetcore-ex は、.NET のデフォルトの .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 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 アプリケーションを追加します。

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

2. アプリケーションによる外部アクセスを可能にします。

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

3. 共有可能な URL を取得します。

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

関連情報

- GitHub の **s2i-dotnetcore-ex** アプリケーションリポジトリー

6.7. 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

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. データベースを追加します。

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

2. .NET アプリケーションを追加します。

```
$ oc new-app dotnet:8.0-ubi8~https://github.com/redhat-developer/s2i-dotnetcore-persistent-ex#dotnet-8.0 --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. アプリケーションによる外部アクセスを可能にします。

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

5. 共有可能な URL を取得します。

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

関連情報

- GitHub の **s2i-dotnetcore-ex** アプリケーションリポジトリー