



.NET 6.0

RHEL 8 で .NET を使い始める

RHEL 8 での .NET 6.0 のインストールおよび実行

.NET 6.0 RHEL 8 で .NET を使い始める

RHEL 8 での .NET 6.0 のインストールおよび実行

法律上の通知

Copyright © 2023 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.

概要

本ガイドでは、RHEL 8 に .NET 6.0 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	3
RED HAT ドキュメントへのフィードバック (英語のみ)	4
第1章 .NET 6.0 の概要	5
第2章 .NET 6.0 のインストール	6
第3章 .NET 6.0 を使用したアプリケーションの作成	7
第4章 .NET 6.0 でのアプリケーションの公開	8
4.1. .NET アプリケーションの公開	8
第5章 コンテナでの .NET 6.0 アプリケーションの実行	10
第6章 OPENSIFT CONTAINER PLATFORM での .NET 6.0 の使用	11
6.1. 概要	11
6.2. .NET イメージストリームのインストール	11
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	14
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	14
6.5. .NET 6.0 の環境変数	15
6.6. MVC サンプルアプリケーションの作成	17
6.7. CRUD サンプルアプリケーションの作成	18
第7章 以前のバージョンの .NET からの移行	20
7.1. 以前のバージョンの .NET からの移行	20
7.2. .NET FRAMEWORK からの移植	20

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](#) を参照してください。

RED HAT ドキュメントへのフィードバック (英語のみ)

弊社ドキュメントに対するご意見をお聞かせください。ドキュメントの改善点があればお知らせください。改善点を報告する場合は、以下のように行います。

- 特定の文章に簡単なコメントを記入する場合は、以下の手順を行います。
 1. ドキュメントの表示が **Multi-page HTML** 形式になっていることを確認してください。ドキュメントの右上隅に **Feedback** ボタンがあることを確認してください。
 2. マウスカーソルを使用して、コメントを追加するテキストの部分を強調表示します。
 3. 強調表示されたテキストの下に表示される **Add Feedback** ポップアップをクリックします。
 4. 表示される指示に従ってください。
- より詳細なフィードバックをお寄せいただく場合は、Bugzilla のチケットを作成してください。
 1. [Bugzilla](#) の Web サイトにアクセスします。
 2. Component として **Documentation** を使用します。
 3. **Description** フィールドに、ドキュメントの改善に向けたご提案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。
 4. **Submit Bug** をクリックします。

第1章 .NET 6.0 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 6.0 は、RHEL 7、RHEL 8、RHEL 9、および OpenShift Container Platform バージョン 3.11 以降でサポートされます。

第2章 .NET 6.0 のインストール

.NET 6.0 は、RHEL 8 の AppStream リポジトリに含まれています。AppStream リポジトリは、RHEL 8 システムでデフォルトで有効になっています。

.NET 6.0 ランタイムは、最新の 6.0 Software Development Kit (SDK) でインストールできます。新しい SDK が .NET 6.0 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 8 をインストールして登録します。詳細は、[標準的な RHEL インストールの実行](#) を参照してください。

手順

- .NET 6.0 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-6.0 -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 6.0 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 6.0 でのアプリケーションの公開

.NET 6.0 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 6.0 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。

- 単一ファイルアプリケーション: アプリケーションは自己完結型であり、1つのバイナリーに依存関係ファイルをすべて含む、単一の実行可能ファイルとしてデプロイできます。
- 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。

- 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。



注記

SCD は IBM Z および LinuxONE では利用できません。

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、[Creating an application using .NET](#) 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 net6.0 -c Release
```

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 net6.0 -c Release -r rhel.8-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**

第5章 コンテナでの .NET 6.0 アプリケーションの実行

ubi8/dotnet-60-runtime イメージを使用して、Linux コンテナ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。

前提条件

- 事前設定されたコンテナ。
以下の例では podman を使用しています。

手順

1. **オプション**: 別のプロジェクトのディレクトリで作業しており、ネストされたプロジェクトの作成を希望しない場合は、プロジェクトの親ディレクトリに戻ります。

```
# cd ..
```

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

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

3. プロジェクトを公開します。

```
$ dotnet publish mvc_runtime_example -f net6.0 -c Release
```

4. **Dockerfile** を作成します。

```
$ cat > Dockerfile <<EOF  
FROM registry.access.redhat.com/ubi8/dotnet-60-runtime  
  
ADD bin/Release/net6.0/publish/ .  
  
CMD ["dotnet", "mvc_runtime_example.dll"]  
EOF
```

5. イメージを構築します。

```
$ podman build -t dotnet-60-runtime-example .
```

6. イメージを実行します。

```
$ podman run -d -p8080:8080 dotnet-60-runtime-example
```

検証手順

- コンテナ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 OPENSIFT CONTAINER PLATFORM での .NET 6.0 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 Life Cycle](#) は、サポートされているバージョンの最新の概要を提供します。

バージョン	タグ	エイリアス
.NET Core 3.1	dotnet:3.1-el7	dotnet:3.1
	dotnet:3.1-ubi8	
.NET 5	dotnet:5.0-ubi8	dotnet:5.0
.NET 6	dotnet:6.0-ubi8	dotnet:6.0

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。

RHEL7 ベース (suffix -el7) は、[registry.redhat.io](#) イメージリポジトリでホストされます。これらのイメージをプルするには、認証が必要です。このような認証情報は、プルシークレットを OpenShift の namespace に追加して設定されます。

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。スクリプトを使用すると、イメージストリームのインストール、更新、または削除を行うことができます。

.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。

6.2.1. OpenShift Client を使用したイメージストリームのインストール

OpenShift クライアント (**oc**) を使用して .NET イメージストリームをインストールできます。

前提条件

- namespace に既存のプルシークレットが存在する。namespace にプルシークレットが存在しない場合は、[Red Hat Container Registry Authentication](#) の手順に従ってこれを追加します。

手順

1. 利用可能な .NET イメージストリームの一覧を表示します。

```
$ oc describe is dotnet
```

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。

- **Error from server (NotFound)** メッセージが表示される場合は、以下を行います。
 - .NET 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.json
```

- **Error from server (NotFound)** メッセージが表示されない場合は、以下を行います。
 - 既存の .NET 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.json
```

6.2.2. Linux および macOS への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、Linux および macOS においてイメージストリームのインストール、アップグレード、または削除を行うことができます。

手順

1. スクリプトをダウンロードします。
 - a. Linux では、以下を使用します。

```
$ wget https://raw.githubusercontent.com/redhat-developer/s2i-dotnetcore/master/install-imagestreams.sh
```

- b. Mac の場合は、以下を使用します。

```
$ curl https://raw.githubusercontent.com/redhat-developer/s2i-dotnetcore/master/install-imagestreams.sh -o install-imagestreams.sh
```

2. スクリプトを実行可能にします。

```
$ chmod +x install-imagestreams.sh
```

3. OpenShift クラスターにログインします。

```
$ oc login
```


4. イメージストリームをインストールし、**registry.redhat.io** に対して認証用のプルシークレットを追加します。

```
./install-imagestreams.sh --os rhel [--user subscription_username --password subscription_password]
```

subscription_username はユーザーの名前に置き換え、**subscription_password** はユーザーのパスワードに置き換えます。RHEL7 ベースのイメージを使用する予定がない場合は、認証情報は省略できます。

プルシークレットが存在する場合は、**--user** と **--password** 引数は無視されます。

追加情報

- **./install-imagestreams.sh --help**

6.2.3. Windows でのイメージストリームのインストール

[このスクリプト](#) を使用すると、Windows のイメージストリームのインストール、アップグレード、または削除を行うことができます。

手順

1. スクリプトをダウンロードします。

```
Invoke-WebRequest https://raw.githubusercontent.com/redhat-developer/s2i-dotnetcore/master/install-imagestreams.ps1 -UseBasicParsing -OutFile install-imagestreams.ps1
```

2. OpenShift クラスターにログインします。

```
$ oc login
```

3. イメージストリームをインストールし、**registry.redhat.io** に対して認証用のプルシークレットを追加します。

```
./install-imagestreams.ps1 --OS rhel [-User subscription_username -Password subscription_password]
```

subscription_username はユーザーの名前に置き換え、**subscription_password** はユーザーのパスワードに置き換えます。RHEL7 ベースのイメージを使用する予定がない場合は、認証情報は省略できます。

プルシークレットがすでに存在する場合は、**-User** と **-Password** 引数は無視されます。



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。

追加情報

- **Get-Help .\install-imagestreams.ps1**

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

次の例では、**oc** を使用して **example-app** アプリケーションをデプロイする方法を示しています。このアプリケーションは、GitHub リポジトリ **redhat-developer/s2i-dotnetcore-ex** の **{dotnet-branch}** ブランチにある **app** フォルダーにあります。

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:6.0-ubi8~https://github.com/redhat-developer/s2i-dotnetcore-ex#{dotnet-branch}' --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、[.NET 6.0 を使用したアプリケーションの公開](#) 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:6.0-ubi8 --binary=true
```

- 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/net6.0/publish
```

- 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 6.0 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の `.s2i/environment` ファイルに追加できます。

変数名	説明	デフォルト
<code>DOTNET_STARTUP_PROJECT</code>	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である必要があります。	.
<code>DOTNET_ASSEMBLY_NAME</code>	実行するアセンブリを選択します。これには <code>.dll</code> 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名 (PropertyGroup/AssemblyName) に設定します。	csproj ファイルの名前
<code>DOTNET_PUBLISH_READYTORUN</code>	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
<code>DOTNET_RESTORE_SOURCES</code>	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。	

変数名	説明	デフォルト
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_SOURCE S 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、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。	

変数名	説明	デフォルト
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	この変数を http://*:8080 に設定して ASP.NET Core がイメージが公開するポートを使用するように設定します。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:6.0-ubi8~https://github.com/redhat-developer/s2i-dotnetcore-ex#  
{dotnet-branch} --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](#)

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:6.0-ubi8~https://github.com/redhat-developer/s2i-dotnetcore-persistent-  
ex#{dotnet-branch} --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](#)

第7章 以前のバージョンの .NET からの移行

7.1. 以前のバージョンの .NET からの移行

Microsoft 社は、ほとんどの以前の .NET Core バージョンから移行する手順を提供します。

サポート対象外になったバージョンの .NET を使用している場合や、新しいバージョンの .NET に移行して機能を拡張する場合は、以下のアートを参照してください。

- [ASP.NET Core 5.0 から 6.0 への移行](#)
- [ASP.NET Core 3.1 から 5.0 への移行](#)
- [ASP.NET Core 3.0 から 3.1 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](#)
- [.NET Core 2.0 から 2.1 への移行](#)
- [ASP.NET から ASP.NET Core への移行](#)
- [project.json からの .NET Core プロジェクトの移行](#)
- [project.json から .csproj 形式への移行](#)



注記

.NET Core 1.x から 2.0 に移行する場合は、[Migrate from ASP.NET Core 1.x to 2.0](#) の最初のいくつかのセクションを参照してください。これらのセクションでは、.NET Core 1.x から 2.0 への移行パスに適した概略を説明しています。

7.2. .NET FRAMEWORK からの移植

.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。
- ライブラリーの移植は、[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。

.NET Framework に存在するいくつかの技術と API は、.NET Core および .NET 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、代替の方法を検討するか、.NET Framework の使用を継続してください。.NET Core および .NET では、次の技術と API はサポートされません。

- Windows Forms や WPF (Windows Presentation Foundation) などのデスクトップアプリケーション
- 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。次の一覧では、この Windows 固有の API の例を示しています。

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



重要

.NET のデフォルトバージョンでサポートされない一部の 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 でのみ動作するため、アプリケーションは Red Hat Enterprise Linux と互換性がありません。