



.NET 3.1

RHEL 7 で .NET を使い始める

RHEL 7 への .NET Core 3.1 のインストールおよび実行

.NET 3.1 RHEL 7 で .NET を使い始める

RHEL 7 への .NET Core 3.1 のインストールおよび実行

Enter your first name here. Enter your surname here.

Enter your organisation's name here. Enter your organisational division here.

Enter your email address here.

法律上の通知

Copyright © 2022 | You need to change the HOLDER entity in the en-US/Getting_started_with_.NET_on_RHEL_7.ent file |.

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 7 に .NET Core 3.1 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の提供	4
第1章 .NET CORE 3.1 の概要	5
第2章 .NET CORE 3.1 のインストール	6
第3章 .NET CORE 3.1 を使用したアプリケーションの作成	8
第4章 .NET CORE 3.1 を使用したアプリケーションの公開	9
4.1. .NET CORE アプリケーションの公開	9
第5章 コンテナでの .NET CORE 3.1 アプリケーションの実行	11
第6章 OPENSIFT CONTAINER PLATFORM での .NET CORE 3.1 の使用	13
6.1. .NET CORE イメージストリームのインストール	13
6.1.1. OpenShift Client を使用したイメージストリームのインストール	13
6.1.2. Linux および macOS へのイメージストリームのインストール	13
6.1.3. Windows でのイメージストリームのインストール	14
6.2. 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	15
6.3. 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	16
6.4. JENKINS スレーブの使用	16
6.5. .NET CORE 3.1 の環境変数	19
6.6. MVC サンプルアプリケーションの作成	21
6.7. CRUD サンプルアプリケーションの作成	22
第7章 .NET の以前のバージョンからの移行	24
7.1. .NET の以前のバージョンからの移行	24
7.2. .NET FRAMEWORK からの移植	24

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、[弊社の CTO、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の提供

ご意見ご要望をお聞かせください。ドキュメントの改善点はございませんか。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** 形式になっていて、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. マウスカーソルで、コメントを追加する部分を強調表示します。
 3. その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. 表示される手順に従ってください。
- より詳細なフィードバックを行う場合は、Bugzilla のチケットを作成します。
 1. [Bugzilla](#) の Web サイトにアクセスします。
 2. Component で **Documentation** を選択します。
 3. **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記入してください。
 4. **Submit Bug** をクリックします。

第1章 .NET CORE 3.1 の概要

.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。.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 Core 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 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 RHEL 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。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 Core 3.1 は、RHEL 7、RHEL 8、RHEL 9、および OpenShift Container Platform バージョン 3.3 以降でサポートされます。

第2章 .NET CORE 3.1 のインストール

RHEL7 に .NET Core をインストールするには、最初に .NET Core ソフトウェアリポジトリを有効にして、**scl** ツールをインストールする必要があります。

前提条件

- 割り当てられたサブスクリプションで RHEL 7 をインストールして登録します。
詳細は、「[システム登録およびサブスクリプションの割り当て](#)」を参照してください。

手順

1. .NET Core ソフトウェアリポジトリを有効にします。

```
$ sudo subscription-manager repos --enable=rhel-7-variant-dotnet-rpms
```

実行している RHEL システム (RHEL 7 Server、RHEL 7 Workstation、または HPC Compute Node) に応じて、**server**、**workstation**、または **hpc-node** に **variant** を置き換えます。

2. システムに割り当てられているサブスクリプションの一覧を確認します。

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

3. **scl** ツールをインストールします。

```
$ sudo yum install scl-utils -y
```

4. .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 rh-dotnet31 -y
```

5. **rh-dotnet31** Software Collection 環境を有効にします。

```
$ scl enable rh-dotnet31 bash
```

この **bash** シェルセッションで **dotnet** コマンドを実行できるようになりました。

ログアウト、別のシェルを使用するか、または新しいターミナルを開くと、**dotnet** コマンドが有効にならなくなります。



警告

Red Hat は、他のプログラムに影響を及ぼす可能性があるため、**rh-dotnet31** を永続的に有効にすることは推奨していません。たとえば、**rh-dotnet31** には、ベースの RHEL バージョン **libcurl** とは異なるバージョンが含まれます。これにより、**libcurl** の別のバージョンのことが予想されないプログラムで問題が発生する可能性があります。**rh-dotnet** を永続的に有効にする場合は、**source scl_source enable rh-dotnet31** を **~/.bashrc** ファイルに追加します。

検証手順

- インストールを確認します。

```
$ dotnet --info
```

出力は、.NET Core インストールおよび環境に関する関連情報を返します。

第3章 .NET CORE 3.1 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。

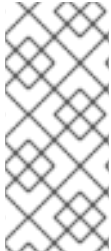
```
Hello World!
```

第4章 .NET CORE 3.1 を使用した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。

.NET Core 3.1 アプリケーションを公開するには、以下のメソッドがあります。

- 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 Core 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。これらのネイティブライブラリーは、**rh-dotnet31** Software Collection に含まれます。

- 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。ネイティブライブラリーが使用できなくなるため、**rh-dotnet31** Software Collection 以外でアプリケーションを実行すると問題が発生する可能性があります。

前提条件

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

4.1. .NET CORE 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 netcoreapp3.1 -c Release
```

my-app を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 restore my-app -r rhel.7-x64  
$ dotnet publish my-app -f netcoreapp3.1 -c Release -r rhel.7-x64 --self-contained false
```

3. Software Collection を有効にし、アプリケーション名を渡して、RHEL システムでアプリケーションを実行します。

```
$ scl enable rh-dotnet31 -- dotnet <app>.dll
```

4. **scl enable rh-dotnet31 — dotnet <app>.dll** コマンドを、アプリケーションで公開されるスクリプトに追加できます。

以下のスクリプトをプロジェクトに追加し、**APP** 変数を更新します。

```
#!/bin/bash

APP=<app>
SCL=rh-dotnet31
DIR="$(dirname "$(readlink -f "$0")")"

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

5. パブリッシュ時にスクリプトを含めるには、この **ItemGroup** を **csproj** ファイルに追加します。

```
<ItemGroup>
  <None Update="<scriptname>" Condition="$(RuntimeIdentifier)' == 'rhel.7-x64' and
'$(SelfContained)' == 'false'" CopyToPublishDirectory="PreserveNewest" />
</ItemGroup>
```

第5章 コンテナでの .NET CORE 3.1 アプリケーションの実行

dotnet/dotnet-31-runtime-rhel7 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ーにいて、ネストされたプロジェクトを作成したくない場合は、プロジェクトの親ディレクトリーに戻ります。

```
$ cd ..
```

2. **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
```

3. プロジェクトを公開します。

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

4. **Dockerfile** を作成します。

```
$ cat > Dockerfile <<EOF
FROM registry.redhat.io/dotnet/dotnet-31-runtime-rhel7

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

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

5. イメージを構築します。

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



注記

unable to retrieve auth token: invalid username/password メッセージを含むエラーが発生した場合は、**registry.redhat.io** サーバーの認証情報を提供する必要があります。**podman login registry.redhat.io** コマンドを使用してログインします。認証情報は通常、Red Hat カスタマーポータルで使用されるものと同じです。

6. イメージを実行します。

```
$ podman run -d -p8080:8080 dotnet-31-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 CORE 3.1 の使用

6.1. .NET CORE 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をインストール、更新、または削除できます。

.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ストリームします。**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。

RHEL 7 イメージストリームを取得するには、サブスクリプション認証情報を使用して **registry.redhat.io** サーバーに対して認証する必要があります。これらのクレデンシャルは、プルシークレットを OpenShift 名前空間に追加して設定されます。

6.1.1. 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 Core イメージストリームをインストールできます。

前提条件

- 名前空間には、既存のプルシークレットが存在する必要があります。名前空間にプルシークレットが存在しない場合、『[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 Core 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。

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.json
```

3. .NET Core イメージストリームがすでにインストールされている場合は、以下を実行して新しいバージョンを含めることができます。

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

6.1.2. 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 にイメージストリームをインストール、アップグレード、または削除できます。

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 rhel7 [--user subscription_username --password subscription_password]
```

subscription_username はユーザーの名前に置き換え、**subscription_password** を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。

関連情報

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

6.1.3. 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.sh --OS rhel7 [-User subscription_username -Password  
subscription_password]
```

subscription_username はユーザーの名前に置き換え、**subscription_password** を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。

関連情報

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

6.2. 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 リポジトリの **dotnetcore-3.1** ブランチの **app** ディレクトリにあります。

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:3.1~https://github.com/redhat-developer/s2i-dotnetcore-ex#dotnetcore-3.1' --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
```

- 共有可能な URL を取得します。

```
$ oc get routes
```

6.3. 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 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。

前提条件

- 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

- 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:3.1 --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/netcoreapp3.1/publish
```

- 新規アプリケーションを作成します。

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

6.4. JENKINS スレーブの使用

OpenShift Container Platform Jenkins イメージは、.NET Core 3.1 スレーブイメージの自動検出を提供します (**dotnet-31**)。

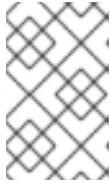
自動検出が機能するには、Jenkins スレーブ **ConfigMap** yaml ファイルをプロジェクトに追加する必要があります。

手順

- Jenkins がデプロイされているプロジェクトに切り替えます。

```
$ oc project _project-name_
```

- dotnet-jenkins-slave.yaml** ファイルを作成します。



注記

<serviceAccount> 要素に使用される値は、Jenkins スレーブによって使用されるアカウントです。値の指定がない場合は、**default** サービスアカウントが使用されます。

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: dotnet-jenkins-slave-31
  labels:
    role: jenkins-slave
data:
  dotnet31: |-
    <org.csanchez.jenkins.plugins.kubernetes.PodTemplate>
    <inheritFrom></inheritFrom>
    <name>dotnet-31</name>
    <instanceCap>2247483647</instanceCap>
    <idleMinutes>0</idleMinutes>
    <label>dotnet-31</label>
    <serviceAccount>jenkins</serviceAccount>
    <nodeSelector></nodeSelector>
    <volumes/>
    <containers>
      <org.csanchez.jenkins.plugins.kubernetes.ContainerTemplate>
        <name>jnlp</name>
        <image>registry.access.redhat.com/dotnet/dotnet-31-jenkins-slave-
rhel7:latest</image>
        <privileged>>false</privileged>
        <alwaysPullImage>true</alwaysPullImage>
        <workingDir>/tmp</workingDir>
        <command></command>
        <args>${computer.jnlpmac} ${computer.name}</args>
        <ttyEnabled>>false</ttyEnabled>
        <resourceRequestCpu></resourceRequestCpu>
        <resourceRequestMemory></resourceRequestMemory>
        <resourceLimitCpu></resourceLimitCpu>
        <resourceLimitMemory></resourceLimitMemory>
        <envVars/>
      </org.csanchez.jenkins.plugins.kubernetes.ContainerTemplate>
    </containers>
    <envVars/>
    <annotations/>
    <imagePullSecrets/>
    <nodeProperties/>
  </org.csanchez.jenkins.plugins.kubernetes.PodTemplate>
```

3. 設定をプロジェクトにインポートします。

```
$ oc create -f dotnet-jenkins-slave.yaml
```

スレーブイメージが使用されるようになりました。

例

以下の例は、OpenShift Container Platform に Jenkins パイプラインを追加する方法を示しています。Jenkins パイプラインが追加され、Jenkins マスターが実行されていない場合、OpenShift はマスターを自動的にデプロイします。Jenkins サーバーインスタンスのデプロイおよび設定に関する詳細は、[OpenShift Container Platform および Jenkins](#) を参照してください。

手順例の **BuildConfig** yaml ファイルには、**dotnet-31** Jenkins スレーブを使用して設定された単純な Jenkins パイプラインが含まれます。サンプル **BuildConfig** yaml ファイルには 3 つの段階があります。

1. ソースがチェックアウトされます。
2. アプリケーションが公開されます。
3. バイナリービルドを使用してイメージがアセンブルされます。
バイナリービルドの詳細については、「[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](#)」を参照してください。

手順

Jenkins マスタースレーブパイプラインを構成するには、以下を行います。

1. **buildconfig.yaml** ファイルを作成します:

```
kind: BuildConfig
apiVersion: v1
metadata:
  name: dotnetapp-build
spec:
  strategy:
    type: JenkinsPipeline
    jenkinsPipelineStrategy:
      jenkinsfile: |-
        node("dotnet-31") {
          stage('clone sources') {
            sh "git clone https://github.com/redhat-developer/s2i-dotnetcore-ex --branch
dotnetcore-3.1 ."
          }
          stage('publish') {
            dir('app') {
              sh "dotnet publish -c Release"
            }
          }
          stage('create image') {
            dir('app') {
              sh 'oc new-build --name=dotnetapp dotnet:3.1 --binary=true || true'
              sh 'oc start-build dotnetapp --from-dir=bin/Release/{buildconfig-var}3.1/publish --
follow'
            }
          }
        }
      }
```

2. **BuildConfig** ファイルを OpenShift にインポートします。

```
$ oc create -f buildconfig.yaml
```

3. OpenShift コンソールを開きます。

4. **Builds → Pipelines** に移動します。
dotnetapp-build パイプラインが利用可能です。
 5. **Start Pipeline** をクリックします。
Jenkins イメージを最初にダウンロードする必要があるため、ビルドを開始するまでに時間がかかる場合があります。
- ビルド時に、OpenShift コンソールでさまざまなパイプラインステージが完了したかどうかを確認できます。**View Log** をクリックし、Jenkins で完了したパイプラインステージを確認することもできます。
6. Jenkins パイプラインのビルドが完了したら、**Builds → Images** に移動します。
dotnetapp イメージがビルドされ、利用可能です。

6.5. .NET CORE 3.1 の環境変数

.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いくつかの環境変数に対応しています。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
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である必要があります。	.
DOTNET_ASSEMBLY_NAME	実行するアセンブリを選択します。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。	

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、ビルドコンテナが CPU 制限の値が低く設定された状態で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 Core 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 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. 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 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. 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 .NET の以前のバージョンからの移行

7.1. .NET の以前のバージョンからの移行

マイクロソフトは、ほとんどの旧バージョンの .NET Core からの移行手順を提供しています。



注記

ASP.NET Core 2.0 から .NET Core 2.1 に移行する場合、次のプロパティは指定しないでください。

<PublishWithAspNetCoreTargetManifest>false</PublishWithAspNetCoreTargetManifest>

.NET Core 2.1 のデフォルト値のままにする必要があります。指定されている場合は、プロジェクトファイルとコマンドラインから必ず削除してください。

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 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ガイダンスを提供しています。

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 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**

[.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 アプリケーションで使用される API が、新しいバージョンの .NET Core 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=<dotnet-framework-version>' --target '.NET Core,Version=<dotnet-version>'
```

<dotnet-framework-version> を、現在使用している .NET Framework のバージョンに置き換えます。たとえば、4.6 です。**<dotnet-version>** を、移行先の .NET Core のバージョンに置き換えます。たとえば、3.1 です。



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。