



NET 2.1

RHEL 7 で .NET を使い始める

ガイド

NET 2.1 RHEL 7 で .NET を使い始める

ガイド

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

Red Hat Enterprise Linux 7 で .NET アプリケーションを開発するための .NET Core 2.1 のインストールおよび実行

目次

RED HAT ドキュメントへのフィードバック (英語のみ)	3
第1章 .NET CORE の概要	4
第2章 RED HAT ENTERPRISE LINUX 7 での .NET CORE 2.1 の使用	5
2.1. .NET CORE 2.1 のインストール	5
2.2. .NET CORE 2.1 を使用したアプリケーションの作成	6
2.3. .NET CORE 2.1 を使用したアプリケーションの公開	7
2.3.1. .NET Core アプリケーションの公開	7
2.3.2. ASP.NET アプリケーションの公開	8
2.4. コンテナでの .NET CORE 2.1 アプリケーションの実行	9
第3章 OPENSIFT CONTAINER PLATFORM での .NET CORE 2.1 の使用	11
3.1. .NET CORE 2.1 イメージストリームのインストール	11
3.2. 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	12
3.3. 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	13
3.4. JENKINS スレーブの使用	14
3.5. .NET CORE 2.1 の環境変数	16
3.6. .NET CORE 2.1 のサンプルアプリケーションの作成	19
第4章 .NET CORE の以前のバージョンからの移行	20
第5章 .NET FRAMEWORK から .NET CORE 2.1 への移行	21
5.1. 移行に関する考慮事項	21
5.2. .NET FRAMEWORK の移行手順	21
付録A 改訂履歴	22

RED HAT ドキュメントへのフィードバック (英語のみ)

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 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 2.1 は、RHEL 7、RHEL 8、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 RED HAT ENTERPRISE LINUX 7 での .NET CORE 2.1 の使用

.NET Core 2.1 をインストールする方法と、.NET Core アプリケーションを作成および公開する方法を説明します。

2.1. .NET CORE 2.1 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 2.1 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-dotnet21 -y
```

5. **rh-dotnet21** Software Collection 環境を有効にします。

```
$ scl enable rh-dotnet21 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-dotnet21** 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-dotnet21** には、ベースの RHEL バージョンとは異なる **libcurl** のバージョンが含まれています。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-dotnet21** 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。

2.2. .NET CORE 2.1 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...
```

```
Restoring packages for /home/username/my-app/my-app.csproj...
```

```
Generating MSBuild file /home/username/my-app/obj/my-app.csproj.nuget.g.props.
```

```
Generating MSBuild file /home/username/my-app/obj/my-app.csproj.nuget.g.targets.
```

```
Restore completed in 224.85 ms for /home/username/my-app/my-app.csproj.
```

```
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。

ディレクトリーには、以下のファイルが含まれます。

```
$ tree my-app
my-app
|__ my-app.csproj
|__ obj
```

```

├── my-app.csproj.nuget.dgspec.json
├── my-app.csproj.nuget.g.props
├── my-app.csproj.nuget.g.targets
├── project.assets.json
├── project.nuget.cache
└── Program.cs

```

1 directory, 7 files

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!
```

2.3. .NET CORE 2.1 を使用したアプリケーションの公開

.NET Core 2.1 アプリケーションを公開して、共有されたシステム全体の .NET Core バージョンを使用するか、.NET Core を追加できます。

.NET Core 2.1 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。RHEL にアプリケーションを公開する場合、Red Hat 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 を使用しているためです。これには、特定のネイティブ依存関係のセットが含まれます。これらのネイティブライブラリーは、**rh-dotnet21** 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利用できないため、**rh-dotnet21** Software Collection 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、[「.NET Core 2.1 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。

2.3.1. .NET Core アプリケーションの公開

手順

- 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 netcoreapp2.1 -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 netcoreapp2.1 -c Release -r rhel.7-x64 --self-contained false
```

3. Software Collection を有効にし、アプリケーションアセンブリ名を dotnet に渡して、RHEL システムでアプリケーションを実行します。

```
$ scl enable rh-dotnet21 -- dotnet <app>.dll
```

4. アプリケーションと共に公開されるスクリプトに **scl enable rh-dotnet21 PROVISIONING-gitopsdotnet <app>.dll** コマンドを追加できます。
以下のスクリプトをプロジェクトに追加し、**ASSEMBLY** 変数を更新します。

```
#!/bin/bash

APP=<app>
SCL=rh-dotnet21
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>
```

2.3.2. ASP.NET アプリケーションの公開

Microsoft SDK を使用する場合、ASP.NET Core 2.1 Web アプリケーションは ASP.NET Core 共有フレームワークの依存関係で公開されます。これは、ランタイムシステムで利用可能であることが予想されるパッケージセットです。

RHEL で公開する場合、これらのパッケージはアプリケーションに含まれます。Microsoft SDK を使用してパッケージを含めるには、以下のようにプロジェクトファイルで **MicrosoftNETPlatformLibrary** プロパティを **Microsoft.NETCore.App** に設定する必要があります。

```
<Project Sdk="Microsoft.NET.Sdk.Web">
  <PropertyGroup>
    <TargetFramework>netcoreapp2.1</TargetFramework>
    <MicrosoftNETPlatformLibrary>Microsoft.NETCore.App</MicrosoftNETPlatformLibrary>
  </PropertyGroup>
  <ItemGroup>
    <PackageReference Include="Microsoft.AspNetCore.App" Version="2.1" />
  </ItemGroup>
</Project>
```

このプロパティは、アプリケーションをパブリッシュするときに設定できます。

```
$ dotnet publish -f netcoreapp2.1 -c Release -r rhel.7-x64 --self-contained false
/p:MicrosoftNETPlatformLibrary=Microsoft.NETCore.App
```

2.4. コンテナでの .NET CORE 2.1 アプリケーションの実行

dotnet/dotnet-21-runtime-rhel7 イメージを使用して、Linux コンテナ内でプリ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. **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 --no-restore
```

2. プロジェクトを復元し、公開します。

```
$ dotnet restore mvc_runtime_example -r rhel.7-x64
$ dotnet publish mvc_runtime_example -f netcoreapp2.1 -c Release -r rhel.7-x64 --self-contained false /p:MicrosoftNETPlatformLibrary=Microsoft.NETCore.App
```

3. **Dockerfile** を作成します。

```
$ cat > Dockerfile <<EOF
FROM registry.redhat.io/dotnet/dotnet-21-runtime-rhel7

ADD bin/Release/netcoreapp2.1/publish/ .

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

4. イメージを構築します。

```
$ podman build -t dotnet-21-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。

5. イメージを実行します。

```
$ podman run -d -p8080:8080 dotnet-21-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3章 OPENSIFT CONTAINER PLATFORM での .NET CORE 2.1 の使用

.NET Core イメージストリームは、Linux、Mac または Windows オペレーティングシステムにインストールできます。

3.1. .NET CORE 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の定義は、**openshift** 名前空間でグローバルに定義することも、特定のプロジェクトでローカルに定義することもできます。

手順

1. システム管理者であるか、または十分なパーミッションがある場合は、**openshift** プロジェクトに切り替えます。**openshift** プロジェクトを使用すると、イメージストリームの定義をグローバルに更新できます。

```
$ oc project openshift
```

openshift プロジェクトを使用するパーミッションがない場合は、手順 2 で始まるプロジェクト定義を更新できます。

2. 利用可能な .NET Core イメージバージョンの一覧を表示します。

```
$ oc describe is dotnet -n openshift
$ 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。

3. イメージをプルするには、OpenShift では **registry.redhat.io** サーバーで認証するための認証情報が必要です。これらの認証情報はシークレットに保存されます。



注記

OpenShift 3.11 以降では、**openshift** namespace 用にシークレットが事前設定されています。

以下のコマンドを入力してシークレットを一覧表示します。最初の列には、シークレット名が表示されます。

```
$ oc get secret | grep kubernetes.io/dockerc
```

シークレットの内容を確認するには、**.dockercfg** または **.dockerconfigjson** データを Base64 形式からデコードできます。これにより、**registry.redhat.io** サーバーの認証情報がすでにあるかどうかを確認できます。以下のコマンドを入力してシークレットの **.dockercfg** セクションを表示します。

```
$ oc get secret <secret-name> -o yaml | grep .dockercfg
.dockercfg:
eyJyZWdpc3RyeS5yZWRoYXQuaW8iOnsidXNlcm5hbWUiOiIqKioqKioqKiIsInBhc3N3b3JkIjojKi
```

```
oqKioqKioiLCJlbWFpbCl6InVudXNIZCIsImF1dGgiOiJLaW9xS2lvcUtpbzZLaW9xS2lvcUtpbz0ifX
0=
```

以下のコマンドに出力をコピーして貼り付けて、Base64 形式から変換します。以下の例は、**registry.redhat.io** サーバーの認証情報を示しています。

```
$ echo
eyJyZWdpc3RyeS5yZWRoYXQuaW8iOmsidXNlcm5hbWUiOiJqKioqKioqKilsInBhc3N3b3JkljoiKi
oqKioqKioiLCJlbWFpbCl6InVudXNIZCIsImF1dGgiOiJLaW9xS2lvcUtpbzZLaW9xS2lvcUtpbz0ifX
0= | base64 -d
{"registry.redhat.io":
{"username":"","password":"","email":"unused","auth":"KioqKioqKio6KioqKioqKio="}}
```

registry.redhat.io サーバーの認証情報と共にシークレットが一覧にない場合は、これを追加する必要があります。

4. Red Hat アカウント認証情報は **registry.redhat.io** アクセス に使用されます。Red Hat 製品のエンタイトルメントをお持ちのお客様は、使用するアカウントの認証情報をお持ちです。通常、これらは Red Hat カスタマーポータルへのログインに使用されるものと同じ認証情報です。Red Hat の認証情報を確認するには、以下のコマンドを実行してログインを試行します。

```
$ podman login registry.redhat.io
```

ログインできない場合は、まず Red Hat のアカウントを取得する必要があります。詳細は、「[Red Hat コンテナレジストリーの認証](#)」を参照してください。ログインできる場合は、以下のコマンドを実行してシークレットを作成します。

```
$ oc create secret docker-registry redhat-registry \
  --docker-server=registry.redhat.io \
  --docker-username=<user-name> \
  --docker-password=<password> \
  --docker-email=unused
$ oc secrets link default redhat-registry --for=pull
$ oc secrets link builder redhat-registry
```

5. 新規イメージストリームをインポートします。

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

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、**replace** コマンドを使用してイメージストリームの定義を更新します。

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

3.2. 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

アプリケーションのデプロイメントに OpenShift クライアント (**oc**) を使用できます。

以下の例は、**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。これは、**redhat-developer/s2i-dotnetcore-ex** GitHub リポジトリの **dotnetcore-2.1** ブランチの **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:2.1~https://github.com/redhat-developer/s2i-dotnetcore-ex#dotnetcore-2.1' --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
```

6. 共有可能な URL を取得します。

```
$ oc get routes
```

3.3. 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

1. 公開済みアプリケーション。
詳細は「[.NET Core 2.1 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:2.1 --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/netcoreapp2.1/publish
```

3. 新規アプリケーションを作成します。

■

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

3.4. JENKINS スレーブの使用

OpenShift Container Platform Jenkins イメージは、.NET Core 2.1 スレーブイメージの自動検出を提供します(dotnet-21)。

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

1. 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_
```

2. **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-21
  labels:
    role: jenkins-slave
data:
  dotnet21: |-
    <org.csanchez.jenkins.plugins.kubernetes.PodTemplate>
    <inheritFrom></inheritFrom>
    <name>dotnet-21</name>
    <instanceCap>2147483647</instanceCap>
    <idleMinutes>0</idleMinutes>
    <label>dotnet-21</label>
    <serviceAccount>jenkins</serviceAccount>
    <nodeSelector></nodeSelector>
    <volumes/>
    <containers>
    <org.csanchez.jenkins.plugins.kubernetes.ContainerTemplate>
    <name>jnlp</name>
    <image>registry.access.redhat.com/dotnet/dotnet-21-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。

この手順例では、**BuildConfig** yaml ファイルには、**dotnet-21** Jenkins スレーブを使用して設定される単純な 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-21") {
          stage('clone sources') {
            sh "git clone https://github.com/redhat-developer/s2i-dotnetcore-ex --branch
dotnetcore-2.1 ."
          }
        }

```

```

stage('publish') {
    dir('app') {
        sh "dotnet publish -c Release
/p:MicrosoftNETPlatformLibrary=Microsoft.NETCore.App"
    }
}
stage('create image') {
    dir('app') {
        sh 'oc new-build --name=dotnetapp dotnet:2.1 --binary=true || true'
        sh 'oc start-build dotnetapp --from-dir=bin/Release/netcoreapp2.1/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。

3.5. .NET CORE 2.1 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。	.

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_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。	
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。	

変数名	説明	デフォルト
<code>DOTNET_RM_SRC</code>	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。	
<code>DOTNET_SSL_DIRS</code>	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。	
<code>NPM_MIRROR</code>	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。	
<code>ASPNETCORE_URLS</code>	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
<code>DOTNET_RESTORE_DISABLE_PARALLEL</code>	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
<code>DOTNET_INCREMENTAL</code>	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。デフォルトは false です。	
<code>DOTNET_PACK</code>	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 <code>/opt/app-root/app.tar.gz</code> に作成されます。	
[OBSOLETE: April 2019] - <code>DOTNET_SDK_VERSION</code>	ビルド時にデフォルトの sdk バージョンを選択します。ソースリポジトリに global.json ファイルがある場合には、これが優先されます。 latest に設定すると、イメージの最新の sdk が使用されます。	イメージで利用可能な最小 sdk バージョン

3.6. .NET CORE 2.1 のサンプルアプリケーションの作成

3つのサンプルアプリケーションが利用できます。

- [dotnet-example](#): これは、デフォルトの model-view-controller(MVC)アプリケーションです。
- [dotnet-runtime-example](#): これは、チェーンビルドを使用して MVC アプリケーションをビルドする方法を示しています。アプリケーションは `dotnet/dotnet-21-rhel7` でビルドされます。結果は `ubi8/dotnet-21-runtime-rhel7` にデプロイされます。チェーンされたビルドは OpenShift Online ではサポートされません。
- [dotnet-pgsql-persistent](#): これは、PostgreSQL バックエンドを使用した Microsoft ASP.NET Core MusicStore サンプルアプリケーションです。

OpenShift Web コンソールを使用してサンプルを追加するには、プロジェクトを参照し、Add to project をクリックします。dotnet のフィルターを設定できます。サンプルが表示されない場合は、以下のコマンドを実行してインストールに追加できます。

```
$ oc create -f https://raw.githubusercontent.com/redhat-developer/s2i-dotnetcore/master/templates/dotnet-example.json
$ oc create -f https://raw.githubusercontent.com/redhat-developer/s2i-dotnetcore/master/templates/dotnet-runtime-example.json
$ oc create -f https://raw.githubusercontent.com/redhat-developer/s2i-dotnetcore/master/templates/dotnet-pgsql-persistent.json
```

第4章 .NET CORE 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。移行時には、以下の ASP.NET Core 2.0 プロパティが指定されなくなります。 .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。

`<PublishWithAspNetCoreTargetManifest>>false</PublishWithAspNetCoreTargetManifest>`

サポートされなくなった .NET Core のバージョンを使用している場合、または新しいバージョンの .NET Core に移行して機能を拡張する場合は、以下の記事を参照してください。

- [.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](#)
- [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。

第5章 .NET FRAMEWORK から .NET CORE 2.1 への移行

以下のセクションを参照して、.NET Framework から移行する方法を説明します。

5.1. 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。

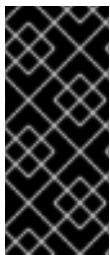
- Windows Forms や WPF (Windows Presentation Foundation) などのデスクトップアプリケーション
- Windows Communication Foundation (WCF) サーバー (WCF クライアントがサポートされています)
- .NET リモート処理

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 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。

5.2. .NET FRAMEWORK の移行手順

.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。

付録A 改訂履歴

日付	バージョン	作成者	変更
08/21/2017	2.0	Les Williams	一般公開
08/30/2017	2.0	Les Williams	セクション 2.3 の DOTNET_STARTUP_PROJECT および DOTNET_TEST_PROJECTS エントリーを修正
09/13/2017	2.0	Les Williams	セクション 1.2 を修正して rh-dotnet20 を永続的に有効にする方法についての注記を追加
02/14/2018	2.0	Les Williams	セクション 2.2 から解決 BZ 1500230; zsh およびその他のシェルの引用を追加
02/28/2018	2.0.3	Les Williams	SDK 2.0 および 2.1 が含まれるように改訂
06/14/2018	2.1	Les Williams	一般公開
08/01/2018	2.1	Toby Drake	移行手順を提供するために第 3 章を追加
08/24/2018	2.1	Toby Drake	ユーザーが新規イメージストリームを取得できるようにする手順を追加
09/18/2018	2.1	Toby Drake	セクション 2.1 を修正。 .NET Core イメージバージョンを一覧表示するコマンドに -n openshift を含めます。検索結果を改善できるように grep コマンドを変更します。
10/12/2018	2.1	Toby Drake	DOTNET_SSL_DIRS および DOTNET_RM_SRC を「 .NET Core 2.1 の環境変数 」に追加。「 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」を追加

日付	バージョン	作成者	変更
11/08/2018	2.1	Toby Drake	docker から podman への参照を変更。レジストリーサーバーを registry.redhat.io に変更しました。Jenkins master-slave パイプラインを設定する手順を追加。「 Jenkins スレーブの使用 」を参照してください。
11/27/2018	2.1	Toby Drake	RHEL 8 に対応するための参照を追加
04/16/2019	2.2	Les Williams	DOTNET_INCREMENTAL および DOTNET_PACK 変数の環境変数セクションを修正