Red Hat build of Quarkus 1.7

Red Hat OpenShift Container Platform での Quarkus アプリケーションのデプロイ

Last Updated: 2021-04-28

Red Hat build of Quarkus 1.7 Red Hat OpenShift Container Platform での Quarkus アプリケーションのデプロイ

法律上の通知

概要

本ガイドでは、Red Hat OpenShift Container Platform に Quarkus アプリケーションをデプロイする方法を説明します。

目次

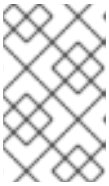
前書き

アプリケーション開発者は、1 つのビルドコマンドで Apache Maven および **quarkus-openshift** エクステンションを使用して、または従来の Source-to-Image (S2I) 手法を使用して、Red Hat OpenShift Container Platform に Quarkus アプリケーションをデプロイすることできます。どちらの手法を使用しても、作成されるイメージは完全にサポートされます。開発サポートの対象となるビルドプロセスおよびツールの詳細は、開発サポートの対象範囲 のページを参照してください。

OpenShift Serverless Knative Serving を使用して、ネイティブ実行可能ファイルにコンパイルされた Quarkus アプリケーションをデプロイし、サービスをスケールアップまたはスケールダウンすることができます。サービスのスケールダウンは、メモリー機能を向上させることができます。

前提条件

- OpenJDK 11 がインストールされ、**JAVA_HOME** 環境変数が Java SDK の場所を指定していること。Red Hat ビルドの Open JDK は、Red Hat カスタマーポータルの Software Downloads ページから入手可能です (ログインが必要です)。
- Apache Maven 3.6.3 以降がインストールされていること。Maven は Apache Maven Project の Web サイトから入手できます。
- Quarkus Maven プロジェクトがあること。Maven を使用した簡単な Quarkus アプリケーションのビルドに関する説明は、**『Quarkus スタートガイド』** を参照してください。



注記

Quarkus Maven プロジェクトの完全なサンプルについては、Quarkus quickstart archive をダウンロードするか、**Quarkus Quickstarts** Git リポジトリーをクローンしてください。この例は **getting-started** ディレクトリーにあります。

- Red Hat OpenShift Container Platform クラスターにアクセスでき、最新バージョンの OpenShift CLI (oc) がインストールされていること。oc のインストールに関する詳細は、**『OpenShift Container Platform クラスターのインストールおよび設定』** ガイドの「CLI のインストール」のセクションを参照してください。

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、弊社 の CTO、Chris Wright のメッセージ を参照してください。

第1章 RED HAT ビルドの QUARKUS

Red Hat ビルドの Quarkus は、コンテナーおよび Red Hat OpenShift Container Platform と使用するために最適化された Kubernetes ネイティブ Java スタックです。Quarkus は、Eclipse MicroProfile、Apache Kafka、RESTEasy (JAX-RS)、Hibernate ORM (JPA)、Spring、Infinispan、Apache Camel などの一般的な Java 標準、フレームワーク、およびライブラリーと連携するように設計されています。

Quarkus のディペンデンシーインジェクション (依存性の注入) ソリューションは、CDI (コンテキストとディペンデンシーインジェクション) をベースとし、エクステンションフレームワークを備えているので、機能の拡張、およびフレームワークの設定、起動、アプリケーションへの統合が可能です。

Quarkus は、コンテナーファーストという手法で Java アプリケーションをビルドします。この手法により、Java で書かれたマイクロサービスベースのアプリケーションのビルドが大幅に容易になるほか、これらのアプリケーションがサーバーレスコンピューティングフレームワークで実行している関数を呼び出すことができるようになります。これにより、Quarkus アプリケーションのメモリーフットプリントは小さくなり、起動時間は高速化されます。

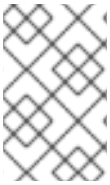
第2章 RED HAT OPENSHIFT CONTAINER PLATFORM

OpenShift Container Platform は、コンテナー化されたアプリケーションを開発し、実行するための Kubernetes ベースのプラットフォームです。アプリケーションおよびアプリケーションをサポートするデータセンターが、わずかなシステムとアプリケーションから、数百万ものクライアントに対応する数千ものシステムへと拡張できるように設計されています。

OpenShift は、アプリケーションのコンテナーイメージをビルドするための 2 つのワークフロー (ソースワークフローとバイナリーワークフロー) をサポートします。どちらのワークフローも Source-to-Image (S2I) 機能をベースとしており、両方のワークフローはソースワークフローを使用して S2I ビルドに依存します。主な違いは、ソースワークフローは OpenShift 内でアプリケーションのデプロイ可能なアーティファクトを生成し、バイナリーワークフローは OpenShift 外でこれらのバイナリーアーティファクトを生成する点です。いずれも OpenShift 内でアプリケーションコンテナーイメージをビルドします。バイナリーワークフローは、コンテナーイメージを生成する OpenShift S2I ビルドのソースとしてアプリケーションバイナリーを提供します。

第3章 QUARKUS OPENSHIFT エクステンションを使用した OPENSHIFT への QUARKUS アプリケーションのデプロイ

従来の Source-to-Image (S2I) ソースワークフローは、OpenShift 内でアプリケーションのデプロイ可能なアーティファクトを生成します。Quarkus OpenShift エクステンションは、S2I バイナリーワークフローを使用して、より効率的なデプロイメントプロセスを提供します。ソースからビルドする代わりに、エクステンションはローカルファイルシステムから JAR ファイルをアップロードします。その結果、ビルドプロセスは従来の S2I 手法よりも最大で 10 倍速くなります。ローカルで開発する場合、およびビルドサーバーまたは継続的インテグレーション (CI) システムから開発する場合に、Quarkus OpenShift エクステンションを使用して、ソースから繰り返し可能なビルドを実行することができます。



注記

Quarkus OpenShift エクステンションは、開発およびテストの目的でのみ使用してください。実稼働環境では、「S2I を使用した OpenShift での Quarkus アプリケーションのデプロイ」で説明されている従来の S2I 手法の使用を検討してください。

手順

1. Quarkus Maven プロジェクトが含まれるディレクトリーに移動します。
2. OpenShift エクステンションを既存プロジェクトに追加するには、以下のコマンドを入力します。

```
./mvnw quarkus:add-extension -Dextensions="openshift"
```

注記

新規プロジェクトの作成時に **-Dextensions="openshift"** 引数を含めて、Quarkus OpenShift エクステンションを追加することができます。

OpenShift エクステンションを追加すると、スクリプトは以下の依存関係を **pom.xml** ファイルに追加します。

```
<dependency>
  <groupId>io.quarkus</groupId>
  <artifactId>quarkus-openshift</artifactId>
</dependency>
```

3. 信頼されていない証明書を使用している場合は、**src/main/resources/application.properties** ファイルに以下の行を追加します。

```
quarkus.kubernetes-client.trust-certs=true
```

4. Open JDK 11 の Red Hat Enterprise Linux 7 イメージを使用するように OpenShift を設定するには、以下の行を **application.properties** ファイルに追加します。

```
quarkus.s2i.base-jvm-image=registry.access.redhat.com/ubi8/openjdk-11
```

注記

IBM Z インフラストラクチャーにデプロイする場合は、**application.properties** ファイルに **quarkus.s2i.base-jvm-image=registry.access.redhat.com/openj9/openj9-11-rhel8** を追加します。

5. OpenShift ルートを作成するには、以下の行を **application.properties** ファイルに追加します。

```
quarkus.openshift.expose=true
```

6. 変更を **application.properties** ファイルに保存します。

7. OpenShift CLI (oc) にログインします。

```
oc login
```

8. 新しい OpenShift プロジェクトを作成するには、以下のコマンドを実行します。このコマンドの **<project_name>** は、新規プロジェクトの名前に置き換えます。

```
oc new-project <project_name>
```

9. プロジェクトを OpenShift にデプロイするには、Quarkus Maven プロジェクトディレクトリーに以下のコマンドを入力します。

```
./mvnw clean package -Dquarkus.kubernetes.deploy=true
```

10. OpenShift プロジェクトでデプロイされたすべてのアプリケーションの名前およびルートを表示するには、以下のコマンドを入力します。

```
oc get route
```

11. アプリケーションの完全な URL を表示するには、以下のコマンドを入力します。ここで、**<application_name>** は OpenShift プロジェクトにデプロイされたアプリケーションの名前になります。

```
export URL="http://$(oc get route <application_name> -o jsonpath='{.spec.host}')"
echo "Application URL: $URL"
curl $URL/hello
```

12. ルートの **hello** エンドポイントに HTTP 要求を実行するには、以下のコマンドを入力します。

```
curl $URL/hello
```

第4章 S2I を使用した OPENSHIFT での QUARKUS アプリケーションのデプロイ

従来の Source-to-Image (S2I) 手法は、Red Hat OpenShift Container Platform でのアプリケーションのデプロイにおける推奨される方法として、広く使用されています。S2I では、Git リポジトリーを使用するか、ビルド時にソースをアップロードして、ソースコードをビルドコンテナーに提供する必要があります。この手法を使用して、実稼働環境に Quarkus アプリケーションをデプロイします。

前提条件

- Git リポジトリーでホストされる Quarkus Maven プロジェクトがあること。

手順

1. Quarkus Maven プロジェクトが含まれるディレクトリーに移動します。
2. **pom.xml** ファイルと同じレベルで、**.s2i** という名前の隠しディレクトリーを作成します。
3. **.s2i** ディレクトリーに **environment** という名前のファイルを作成し、以下の内容を追加します。

```
ARTIFACT_COPY_ARGS=-p -r lib/ *-runner.jar
```

4. 変更をリモート Git リポジトリーにコミットし、プッシュします。
5. OpenShift CLI (oc) にログインします。

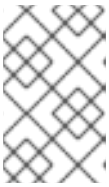
```
oc login
```

6. 新しい OpenShift プロジェクトを作成するには、以下のコマンドを実行します。このコマンドの **<project_name>** は、新規プロジェクトの名前に置き換えます。

```
oc new-project <project_name>
```

7. サポートされる OpenShift イメージをインポートするには、以下のコマンドを入力します。

```
oc import-image --confirm ubi8/openjdk-11 --from=registry.access.redhat.com/ubi8/openjdk-11
```



注記

IBM Z インフラストラクチャーにデプロイする場合は、**oc import-image --confirm openj9/openj9-11-rhel8 --from=registry.redhat.io/openj9/openj9-11-rhel8** を入力します。

このイメージの詳細は、Red Hat OpenJ9 11 Java Applications on RHEL8 ページを参照してください。

8. OpenShift でプロジェクトをビルドするには、以下のコマンドを入力します。**<git_path>** は Quarkus プロジェクトをホストする Git リポジトリーへのパス、**<project_name>** は作成した OpenShift プロジェクトに置き換えます。

```
oc new-app ubi8/openjdk-11 <git_path> --name=<project_name>
```

注記

IBM Z インフラストラクチャーにデプロイする場合は、**oc new-app openj9/openj9-11-rhel8 <git_path> --name=<project_name>** を入力します。

このコマンドを実行すると、プロジェクトのビルド、アプリケーションの作成、そして OpenShift サービスのデプロイが行われます。

9. OpenShift ルートを作成するには、以下のコマンドを入力します。

```
oc expose service/<project_name>
```

10. 新規ルートを表示するには、以下のコマンドを入力します。このコマンドの <application_name> は、OpenShift プロジェクトにデプロイされたアプリケーションの名前に置き換えます。

```
export URL="http://$(oc get route <application_name> -o jsonpath='{.spec.host}}')"
echo "Application URL: $URL"
```

11. ルートの **hello** エンドポイントに HTTP 要求を実行するには、以下のコマンドを入力します。

```
curl $URL/hello
```

12. アプリケーションを使用するには、前のコマンドで返された URL を Web ブラウザーで入力します。

13. プロジェクトの更新バージョンをデプロイするには、更新を Git リポジトリーにプッシュしてから、以下のコマンドを入力します。

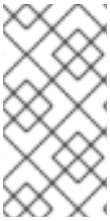
```
oc start-build <project_name>
```

14. ビルドの完了後にブラウザーページを更新し、変更を確認します。

第5章 OPENSHIFT SERVERLESS サービスとしてネイティブ実行可能ファイルにコンパイルされた QUARKUS アプリケーションのデプロイ

アプリケーション開発者は、OpenShift Serverless Knative Serving を使用して、ネイティブ実行可能ファイルにコンパイルされた Quarkus アプリケーションを Red Hat OpenShift Container Platform にデプロイできます。

OpenShift Serverless Knative Serving を使用することで、読み込むサイズに応じてサービスをスケールアップまたはスケールダウンすることができます。現在リクエストされていないサービスをスケールダウンすることで、メモリー機能を向上させることができます。



注記

Quarkus をネイティブ実行可能ファイルとして、または OpenJDK を使用する Java アプリケーションとして実行することができます。ネイティブ実行可能ファイルの場合は、Red Hat UBI 8 minimal イメージを使用します。OpenJDK の場合は、Red Hat 8 UBI Java イメージを使用します。

5.1. サーバーレスアプリケーションとして、継続的インテグレーションで QUARKUS ネイティブアプリケーションのコンテナーイメージをデプロイします。

ネイティブサーバーレスアプリケーションをデプロイする場合は、ネイティブビルド、コンテナービルド、およびデプロイメントの手順を別々にすることができます。以下の手順では、サーバーレスアプリケーションとして継続的インテグレーション (CI) で Quarkus ネイティブアプリケーションのコンテナーイメージをデプロイする方法を説明しています。

前提条件

- OpenShift Serverless Operator がインストールされていること。
- OpenShift Knative Serving がインストールされていること。
- ネイティブコンパイルの場合は、Podman または Docker などのコンテナー環境が必要。
- **kn** CLI ツールがインストールされていること。

手順

1. Quarkus プロジェクトが含まれるディレクトリーに移動します。
2. 以下のいずれかの方法を使用して、Linux 実行可能ファイルをビルドします。
   a. Docker を使用する場合

```
./mvnw package -Pnative -Dquarkus.native.container-build=true -
Dquarkus.native.builder-image=registry.access.redhat.com/quarkus/mandrel-20-
rhel8:20.3
```

   b. Podman を使用する場合

```
./mvnw package -Pnative -Dquarkus.native.container-build=true -
Dquarkus.native.container-runtime=podman -Dquarkus.native.builder-
image=registry.access.redhat.com/quarkus/mandrel-20-rhel8:20.3
```

3. **src/main/docker/Dockerfile.native** ファイルを開き、**<image_name>** パラメーターおよび **<version>** パラメーターを設定します。

   a. Docker を使用する場合

   ```
   docker build -f src/main/docker/Dockerfile.native -t <image_name>:<version> .
   ```

   b. Podman を使用する場合

   ```
   podman build -f src/main/docker/Dockerfile.native -t <image_name>:<version>.
   ```

4. CI 環境および OpenShift 環境がアクセスできるリポジトリーにコンテナーをプッシュします。ここで、**<registry>** はレジストリーの URL になります。

   a. Docker を使用する場合

   ```
   docker tag <image_name>:<version> <registry>/<image_name>:<version>
   docker push <registry>/<image_name>:<version>
   ```

   b. Podman を使用する場合

   ```
   podman tag <image_name>:<version> <registry>/<image_name>:<version>
   podman push <registry>/<image_name>:<version>
   ```

5. OpenShift CLI (oc) にログインします。

   ```
   oc login
   ```

6. 新しい OpenShift プロジェクトを作成するには、以下のコマンドを実行します。このコマンドの **<project_name>** は、新規プロジェクトの名前に置き換えます。

   ```
   oc new-project <project_name>
   ```

7. OpenShift Serverless CLI (kn) を使用して、サーバーレスアプリケーションとしてコンテナーをデプロイするには、以下のコマンドを入力します。**<service_name>** は、お使いのサービスの名前に置き換えます。

   ```
   kn service create <service_name> --image REPOSITORY/<image_name>:<version>
   ```

8. サービスの準備ができていることを確認するには、以下のコマンドを入力します。

   ```
   kn service list <service_name>
   ```

   サービスの準備ができている場合は、「READY」と呼ばれる列の出力に **true** と表示されます。

注記

イメージがプルダウンされて準備ができたときではなく、必要なコンポーネントが作成されるときに、**kn service** コマンドは **true** を返します。

第6章 その他のリソース

- Maven を使用した Quarkus アプリケーションの作成に関する詳細は、**『Apache Maven を使用した Quarkus アプリケーションの開発およびコンパイル』** を参照してください。
- Quarkus アプリケーションをネイティブ実行可能ファイルにコンパイルする方法や、ネイティブ実行可能ファイルのテストに関する詳細は、**『Quarkus アプリケーションのネイティブ実行可能ファイルへのコンパイル』** を参照してください。

改訂日時: 2021-04-27 03:43:49 UTC