Red Hat build of Node.js 12

『Node.js Runtime Guide』

Node.js 12 を使用して、OpenShift およびスタンドアロン RHEL 上で動作するスケーラブルなネットワークアプリケーションを開発します。

Last Updated: 2021-04-03

Red Hat build of Node.js 12 『Node.js Runtime Guide』

Node.js 12 を使用して、OpenShift およびスタンドアロン RHEL 上で動作するスケーラブルなネットワークアプリケーションを開発します。

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

概要

本ガイドでは、Node.js ランタイムの使用方法について説明します。

目次

PREFACE

本ガイドでは、概念と、開発者が Node.js ランタイムを使用するために必要な実用的な詳細を説明します。

第1章 NODE.JS を使用したアプリケーション開発の概要

本セクションでは、Red Hat ランタイムを使用したアプリケーション開発の基本概念について説明します。また、Node.js ランタイムの概要も提供します。

1.1. RED HAT RUNTIMES でのアプリケーション開発の概要

Red Hat OpenShift は、クラウドネイティブランタイムのコレクションを提供するコンテナーアプリケーションプラットフォームです。ランタイムを使用して、OpenShift で Java または JavaScript アプリケーションを開発、ビルド、およびデプロイできます。

Red Hat Runtimes for OpenShift を使用したアプリケーションの開発には、以下が含まれます。

- OpenShift 上で実行されるよう設計された Eclipse Vert.x、Thorntail、Spring Boot などのランタイムのコレクション。
- OpenShift 上のクラウドネイティブな開発に対する規定的なアプローチ。

OpenShift は、アプリケーションのデプロイメントおよびモニタリングを管理、保護、および自動化するのに役立ちます。ビジネス問題を小さなマイクロサービスに分割し、OpenShift を使用してマイクロサービスをデプロイ、監視、および維持できます。サーキットブレーカー、ヘルスチェック、サービス検出などのパターンをアプリケーションに実装できます。

クラウドネイティブ開発は、クラウドコンピューティングをフル活用します。

以下でアプリケーションをビルド、デプロイ、および管理することができます。

OpenShift Container Platform

Red Hat によるオンプレミスクラウド

Red Hat Container Development Kit(Minishift)

ローカルマシンでインストールおよび実行できるローカルクラウド。この機能は、Red Hat Container Development Kit (CDK) または Minishift で提供されます。

Red Hat CodeReady Studio

アプリケーションの開発、テスト、およびデプロイのための統合開発環境(IDE)。

アプリケーションの開発を容易にするため、すべてのランタイムをサンプルアプリケーションで使用できます。これらのサンプルアプリケーションは、Developer Launcher からアクセスできます。サンプルをテンプレートとして使用し、アプリケーションを作成できます。

本ガイドでは、Node.js ランタイムについての詳細情報を提供します。その他のランタイムの詳細は、関連する ランタイムドキュメント を参照してください。

1.2. DEVELOPER LAUNCHER を使用した RED HAT OPENSHIFT でのアプリケーション開発

Developer Launcher (developers.redhat.com/launch) を使用して、OpenShift でのクラウドネイティブアプリケーションの開発を開始することができます。これは、Red Hat が提供するサービスです。

Developer Launcher はスタンドアロンのプロジェクトジェネレーターです。これを使用すると、OpenShift Container Platform、Minishift、または CDK などの OpenShift インスタンスにアプリケーションをビルドおよびデプロイできます。

1.3. NODE.JS の概要

Node.js は Google の V8 JavaScript エンジン をベースとしており、サーバー側の JavaScript アプリケーションを作成できます。イベントおよび非ブロッキング操作をベースとした I/O モデルを提供し、効率的なアプリケーションを作成できます。Node.js は 、npm と呼ばれる大規模なモジュールエコシステムも提供します。Node.js に関する詳細は、追加のリソース を確認してください。

Node.js ランタイムを使用すると、ローリングアップデート、継続的デリバリーパイプライン、サービス検出、カナリアデプロイメントなどの OpenShift プラットフォームの利点や利便性をすべて提供し、Node.js アプリケーションおよびサービスおよびサービスを実行できます。また、OpenShift では、外部化設定、ヘルスチェック、サーキットブレーカー、フェイルオーバーなどの一般的なマイクロサービスパターンを実装することがより簡単になります。

Red Hat は、サポートされるさまざまな Node.js リリースを提供します。サポートの利用方法については、「 Getting Node.js and support from Red Hat 」を参照してください。

1.3.1. Node.js でサポートされるアーキテクチャー

Node.js は以下のアーキテクチャーをサポートします。

- x86_64 (AMD64)
- OpenShift 環境の IBM Z(s390x)

アーキテクチャーごとに異なるイメージがサポートされます。本ガイドのコード例は、x86_64 アーキテクチャーのコマンドを示しています。他のアーキテクチャーを使用している場合は、コマンドに適切なイメージ名を指定します。

1.3.2. サンプルアプリケーションの概要

例としては、クラウドネイティブのアプリケーションとサービスを構築する方法を実証する作業用アプリケーションが挙げられます。これらは、アプリケーションの開発時に使用する必要がある規定アーキテクチャー、設計パターン、ツール、およびベストプラクティスを示しています。サンプルアプリケーションは、クラウドネイティブマイクロサービスを作成するためのテンプレートとして使用できます。本ガイドで説明されているデプロイメントプロセスを使用して、この例を更新し、再デプロイすることができます。

この例では、以下のような マイクロサービスパターン を実装します。

- REST API の作成
- データベースとの相互運用性
- ヘルスチェックパターンの実装
- アプリケーションの設定の外部化により、よりセキュアでスケーリングが容易になります。

サンプルアプリケーションは、以下のように使用できます。

- 技術のデモンストレーション
- プロジェクトのアプリケーションを開発する方法を理解するためのツールまたはサンドボックスを学習する
- 独自のユースケースの更新または拡張の開始

各サンプルアプリケーションは、1つ以上のランタイムに実装されます。たとえば、REST API Level 0 の例は、以下のランタイムで利用できます。

- Node.js
- Spring Boot
- Eclipse Vert.x
- Thorntail

以降のセクションでは、Node.js ランタイムに実装されたサンプルアプリケーションについて説明します。

以下のサンプルアプリケーションをすべてダウンロードしてデプロイできます。

- x86_64 アーキテクチャー： 本ガイドのアプリケーションの例は、x86_64 アーキテクチャーでサンプルアプリケーションをビルドおよびデプロイする方法を説明します。
- s390x アーキテクチャー： IBM Z インフラストラクチャーでプロビジョニングされる OpenShift 環境にサンプルアプリケーションをデプロイするには、コマンドに適切な IBM Z イメージ名を指定します。
  このサンプルアプリケーションでは、ワークフローを実証するために Red Hat Data Grid などの他の製品も必要です。この場合、これらの製品のイメージ名を、サンプルアプリケーションの YAML ファイルの関連する IBM Z イメージ名に変更する必要があります。

第2章 DEVELOPER LAUNCHER を使用したアプリケーションのダウンロードおよびデプロイ

ここでは、ランタイムで提供されるサンプルアプリケーションをダウンロードしてデプロイする方法を説明します。サンプルアプリケーションは Developer Launcher で利用できます。

2.1. DEVELOPER LAUNCHER の使用

Developer Launcher (developers.redhat.com/launch) は OpenShift 上で実行します。サンプルアプリケーションをデプロイする場合、Developer Launcher は以下のプロセスについて説明します。

- ランタイムの選択
- アプリケーションのビルドおよび実行

選択した内容に基づいて、Developer Launcher はカスタムプロジェクトを生成します。ZIP バージョンをダウンロードするか、または OpenShift Online インスタンス上でアプリケーションを直接起動できます。

Developer Launcher を使用してアプリケーションを OpenShift にデプロイする場合は、Source-to-Image (S2I) ビルドプロセスが使用されます。このビルドプロセスは、OpenShift でアプリケーションを実行するために必要なすべての設定、ビルド、およびデプロイメントの手順を処理します。

2.2. DEVELOPER LAUNCHER を使用したサンプルアプリケーションのダウンロード

Red Hat は、Node.js ランタイムの使用を開始するのに役立つサンプルアプリケーションを提供しています。これらの例は、Developer Launcher (developers.redhat.com/launch) で利用できます。

サンプルアプリケーションをダウンロードしてビルドし、デプロイできます。本セクションでは、サンプルアプリケーションのダウンロード方法を説明します。

サンプルアプリケーションをテンプレートとして使用し、独自のクラウドネイティブアプリケーションを作成できます。

手順

1. Developer Launcher (developers.redhat.com/launch) に移動します。
2. **Start をクリックし** ます。
3. **Deploy an Example Application**をクリックします。
4. **Select an Example** をクリックし、ランタイムで使用できるサンプルアプリケーションの一覧を表示します。
5. ランタイムを選択します。
6. サンプルアプリケーションを選択します。

注記

アプリケーションの例は、複数のランタイムで利用できます。直前の手順でランタイムを選択していない場合は、サンプルアプリケーションで利用可能なランタイムの一覧からランタイムを選択できます。

7. ランタイムのリリースバージョンを選択します。ランタイムに一覧表示されているコミュニティーまたは製品リリースを選択できます。
8. **Save をクリックし** ます。
9. **Download** をクリックしてサンプルアプリケーションをダウンロードします。
ソースおよびドキュメントファイルが含まれる ZIP ファイルがダウンロードされている。

2.3. OPENSHIFT CONTAINER PLATFORM または CDK(MINISHIFT)へのサンプルアプリケーションのデプロイ

サンプルアプリケーションは、OpenShift Container Platform または CDK(Minishift)のいずれかにデプロイできます。アプリケーションのデプロイ先に応じて、認証に該当する Web コンソールが使用されます。

前提条件

- Developer Launcher を使用して、サンプルアプリケーションプロジェクトを作成している。
- アプリケーションを OpenShift Container Platform にデプロイする場合、OpenShift Container Platform Web コンソールにアクセスできる必要があります。
- CDK(Minishift)にアプリケーションをデプロイする場合は、CDK(Minishift)Web コンソールにアクセスできる必要があります。
- **OC** コマンドラインクライアントがインストールされている。

手順

1. サンプルアプリケーションをダウンロードします。
2. **oc** コマンドラインクライアントを使用して、サンプルアプリケーションを OpenShift Container Platform または CDK(Minishift)にデプロイできます。
Web コンソールが提供するトークンを使用してクライアントを認証する必要があります。アプリケーションのデプロイ先に応じて、OpenShift Container Platform Web コンソールまたは CDK(Minishift)Web コンソールのいずれかを使用します。クライアントを認証するには、以下の手順を実行します。
   a. Web コンソールにログインします。
   b. Web コンソールの右上隅にある疑問符アイコンをクリックします。
   c. 一覧から **Command Line Tools** を選択します。
   d. **oc login** コマンドをコピーします。
   e. ターミナルにコマンドを貼り付けます。**oc** CLI クライアントをアカウントで認証します。

```
$ oc login OPENSHIFT_URL --token=MYTOKEN
```

3. ZIP ファイルの内容を展開します。

```
$ unzip MY_APPLICATION_NAME.zip
```

4. OpenShift で新規プロジェクトを作成します。

```
$ oc new-project MY_PROJECT_NAME
```

5. **MY_APPLICATION_NAME のルートディレクトリーに移動し** ます。

6. **npm** を使用して OpenShift へのデプロイメントを開始します。

```
$ npm install && npm run openshift
```

これらのコマンドは、不足しているモジュール依存関係をすべてインストールし、Nodeshift モジュールを使用してサンプルアプリケーションを OpenShift にデプロイします。

注記： アプリケーションの例によっては、追加の設定が必要になる場合があります。サンプルアプリケーションをビルドおよびデプロイするには、**README** ファイルに記載されている手順に従います。

7. アプリケーションのステータスを確認し、Pod が実行されていることを確認します。

```
$ oc get pods -w
NAME                      READY     STATUS      RESTARTS   AGE
MY_APP_NAME-1-aaaaa             1/1       Running     0          58s
MY_APP_NAME-s2i-1-build         0/1       Completed   0          2m
```

MY_APP_NAME-1-aaaaa Pod の完全なデプロイおよび起動後にステータスが **Running** になります。アプリケーションの Pod 名は異なる場合があります。Pod 名の数値は新規ビルドごとに増分されます。末尾の文字は、Pod の作成時に生成されます。

8. サンプルアプリケーションがデプロイされ、起動したら、そのルートを決定します。

ルート情報の例

```
$ oc get routes
NAME              HOST/PORT                                        PATH      SERVICES
PORT      TERMINATION
MY_APP_NAME       MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME
MY_APP_NAME      8080
```

Pod のルート情報は、アクセスに使用できるベース URL を提供します。この例では、http://MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME をベース **URL** として使用してアプリケーションにアクセスできます。

第3章 NODE.JS アプリケーションの開発およびデプロイ

この 例を使用する 他に、新規 Node.js アプリケーションをゼロから OpenShift にデプロイできます。

3.1. NODE.JS アプリケーションの開発

基本的な Node.js アプリケーションの場合、Node.js メソッドが含まれる JavaScript ファイルを作成する必要があります。

前提条件

- **npm** がインストールされている。

手順

1. 新しいディレクトリー myApp を作成 し、そのディレクトリーに移動します。

```
$ mkdir myApp
$ cd MyApp
```

これは、アプリケーションのルートディレクトリーです。

2. **npm** でアプリケーションを初期化します。
この例では、エントリーポイントが **app.js** であることを仮定し 、**npm init** の実行時に設定するようにプロンプトが表示されます。

```
$ cd myApp
$ npm init
```

3. **app.js** という新規ファイルにエントリーポイントを作成します。

app.jsの例

```
const http = require('http');

const server = http.createServer((request, response) => {
  response.statusCode = 200;
  response.setHeader('Content-Type', 'application/json');

  const greeting = {content: 'Hello, World!'};

  response.write(JSON.stringify(greeting));
  response.end();
});

server.listen(8080, () => {
  console.log('Server running at http://localhost:8080');
});
```

4. アプリケーションを起動します。

```
$ node app.js
Server running at http://localhost:8080
```

5. **curl** またはブラウザーを使用して、アプリケーションが **http://localhost:8080** で稼働していることを確認します。

```
$ curl http://localhost:8080
{"content":"Hello, World!"}
```

追加情報

- Node.js ランタイムは、Node.js API ドキュメントに記載されているコア Node.js API を提供します。

3.2. NODE.JS アプリケーションの OPENSHIFT へのデプロイ

Node.js アプリケーションを OpenShift にデプロイするには、nodeshift をアプリケーション **に** 追加し、**package.json** ファイルを設定してから、nodeshift を使用してデプロイし **ます**。

3.2.1. OpenShift デプロイメント向けの Node.js アプリケーションの準備

OpenShift デプロイメントの Node.js アプリケーションを準備するには、以下の手順を実行する必要があります。

- nodeshift を **アプリケーション** に追加します。
- **openshift** および **start** エントリーを **package.json** ファイルに追加します。

前提条件

- **npm** がインストールされている。

手順

1. nodeshift を **アプリケーション** に追加します。

```
$ npm install nodeshift --save-dev
```

2. **openshift** および **start** エントリーを **package.json** の **scripts** セクションに追加します。

```
{
  "name": "myApp",
  "version": "1.0.0",
  "description": "",
  "main": "app.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "openshift": "nodeshift --expose --
dockerImage=registry.access.redhat.com/rhscl/ubi8/nodejs-12",
    "start": "node app.js",
    ...
  }
  ...
}
```

openshift スクリプトは nodeshift を使用 **し** てアプリケーションを OpenShift にデプロイします。

注記

汎用ベースイメージと RHEL イメージは Node.js で利用できます。イメージ名の詳細は、『Node.js リリースノート』を参照してください。

3. **オプション**: **package.json** に **files** セクションを追加します。

```
{
  "name": "myApp",
  "version": "1.0.0",
  "description": "",
  "main": "app.js",
  "scripts": {
    ...
  },
  "files": [
    "package.json",
    "app.js"
  ]
  ...
}
```

files セクションは、OpenShift にデプロイする際に追加する **ファイル** とディレクトリーを nodeshift に指示します。**nodeshift** は **node-tar** モジュールを使用して、files セクションに一覧表示されているファイルおよびディレクトリーに基づいて tar **ファイル** を作成します。この tar ファイルは、nodeshift がアプリケーションを OpenShift **に** デプロイする場合に使用されます。**files** セクションが指定されていない場合、nodeshift **は** 以下を除き、現在のディレクトリー全体を送信します。

- **node_modules/**
- **.git/**
- **tmp/**
  OpenShift へのデプロイ時に不要な **ファイル** を含めないように、**package.json** に files セクションを追加することが推奨されます。

3.2.2. Node.js アプリケーションの OpenShift へのデプロイ

nodeshift を使用して Node.js アプリケーションを OpenShift にデプロイでき **ます**。

前提条件

- **oc** CLI クライアントがインストールされていること。
- **npm** がインストールされている。
- ルートの設定時にアプリケーションによって使用されるすべてのポートが正しく公開されることを確認します。

手順

1. oc クライアントで OpenShift インスタンスに **ログイン** します。

```
$ oc login ...
```

2. nodeshift を **使用** して、アプリケーションを OpenShift にデプロイします。

```
$ npm run openshift
```

3.3. NODE.JS アプリケーションのスタンドアロン RED HAT ENTERPRISE LINUX へのデプロイ

スタンドアロンの Red Hat Enterprise Linux に、**npm** を使用して、Node.js アプリケーションをデプロイすることができます。

前提条件

- Node.js アプリケーション。
- npm 6.4.1 installed
- RHEL 7 または RHEL 8 がインストールされている。
- Node.js がインストールされている。

手順

1. プロジェクトの **package.json** ファイルに追加の依存関係を指定した場合は、アプリケーションを実行する前にインストールしてください。

```
$ npm install
```

2. アプリケーションの root ディレクトリーからアプリケーションをデプロイします。

```
$ node app.js
Server running at http://localhost:8080
```

検証手順

1. **curl** またはブラウザーを使用して、アプリケーションが **http://localhost:8080**で実行されていることを確認します。

```
$ curl http://localhost:8080
```

第4章 NODE.JS ベースのアプリケーションのデバッグ

本セクションでは、Node.js ベースのアプリケーションをデバッグし、ローカルデプロイメントとリモートデプロイメントの両方でデバッグロギングを使用する方法について説明します。

4.1. リモートデバッグ

アプリケーションをリモートでデバッグするには、デバッグモードで起動し、デバッガーを割り当てる必要があります。

4.1.1. アプリケーションのローカル起動およびネイティブデバッガーの割り当て

ネイティブデバッガーを使用すると、組み込みのデバッグクライアントを使用して Node.js ベースのアプリケーションをデバッグできます。

前提条件

- デバッグするアプリケーション。

手順

1. デバッガーを有効にしてアプリケーションを起動します。
   ネイティブデバッガーが自動的に割り当てられ、デバッグプロンプトが表示されます。

   デバッガーが有効なアプリケーションの例

   ```
   $ node inspect app.js
   < Debugger listening on ws://127.0.0.1:9229/12345678-aaaa-bbbb-cccc-0123456789ab
   < For help see https://nodejs.org/en/docs/inspector
   < Debugger attached.
   ...
   debug>
   ```

   アプリケーションのエントリーポイントが異なる場合は、コマンドを変更して、そのエントリーポイントを指定する必要があります。

   ```
   $ node inspect path/to/entrypoint
   ```

   たとえば、表現ジェネレーター を使用してアプリケーションを作成すると、エントリーポイントはデフォルトで **./bin/www** に設定されます。REST API レベル 0 のサンプルアプリケーションなど、一部の例 では、**./bin/www** をエントリーポイントとして使用します。

2. Debuginfo プロンプトを使用して デバッグコマンド を実行します。

4.1.2. アプリケーションのローカル起動および V8 ファクトリーのアタッチ

V8 inspector を使用すると、Chrome Debugging Protocol を使用する Chrome DevTools などの他のツールを使用して Node.js ベースのアプリケーションをデバッグ できます。

前提条件

- デバッグするアプリケーション。

- Google Chrome ブラウザーで提供されるものなど、V8 ウォーターがインストールさ れました。

手順

1. V8 inspector 統合を有効 にしてアプリケーションを起動します。

```
$ node --inspect app.js
```

アプリケーションのエントリーポイントが異なる場合は、コマンドを変更して、そのエントリーポイントを指定する必要があります。

```
$ node --inspect path/to/entrypoint
```

たとえば、表現ジェネレーター を使用してアプリケーションを作成すると、エントリーポイントはデフォルトで **./bin/www** に設定されます。REST API レベル 0 のサンプルアプリケーションなど、一部の例 では、**./bin/www** をエントリーポイントとして使用します。

2. V8 inspector をアタッチし、デバッグコマンドを実行します。
たとえば、Google Chrome を使用している場合は、以下のようになります。

   a. **chrome://inspect に移動し** ます。

   b. **Remote Target** の下でアプリケーションを選択します。

   c. アプリケーションのソースが表示され、デバッグアクションを実行できるようになりました。

4.1.3. デバッグモードでの OpenShift でのアプリケーションの起動

OpenShift の Node.js ベースのアプリケーションをリモートでデバッグするには、コンテナー内の **NODE_ENV** 環境変数を **開発** に設定し、ポート転送を設定して、リモートデバッガーからアプリケーションに接続できるようにする必要があります。

前提条件

- OpenShift で実行しているアプリケーション。
- **oc** バイナリーがインストールされている。
- ターゲット OpenShift 環境で **oc port-forward** コマンドを実行する機能。

手順

1. **oc** コマンドを使用して、利用可能なデプロイメント設定を一覧表示します。

```
$ oc get dc
```

2. アプリケーションのデプロイメント設定で **NODE_ENV** 環境変数を **development** に設定してデバッグを有効にします。以下に例を示します。

```
$ oc set env dc/MY_APP_NAME NODE_ENV=development
```

3. 設定変更時に自動的に再デプロイするよう設定されていなければ、アプリケーションを再デプロイします。以下に例を示します。

```
$ oc rollout latest dc/MY_APP_NAME
```

4. ローカルマシンからアプリケーション Pod へのポート転送を設定します。

   a. 現在実行中の Pod を一覧表示し、アプリケーションが含まれるものを見つけます。

```
$ oc get pod
NAME                        READY     STATUS      RESTARTS   AGE
MY_APP_NAME-3-1xrsp         0/1       Running     0          6s
...
```

   b. ポート転送を設定します。

```
$ oc port-forward MY_APP_NAME-3-1xrsp $LOCAL_PORT_NUMBER:5858
```

   ここで、**$LOCAL_PORT_NUMBER** は、ローカルマシンの任意の未使用ポート番号です。リモートデバッガー設定については、この数字を忘れないでください。

5. V8 inspector をアタッチし、デバッグコマンドを実行します。
   たとえば、Google Chrome を使用している場合は、以下のようになります。

   a. **chrome://inspect に移動し** ます。

   b. **設定を** クリックします。

   c. 127.0.0.1:$LOCAL_PORT_NUMBER を追加 **し** ます。

   d. **完了をクリックし** ます。

   e. **Remote Target** の下でアプリケーションを選択します。

   f. アプリケーションのソースが表示され、デバッグアクションを実行できるようになりました。

6. デバッグが完了したら、アプリケーション Pod で **NODE_ENV** 環境変数の設定を解除します。以下に例を示します。

```
$ oc set env dc/MY_APP_NAME NODE_ENV-
```

4.2. デバッグロギング

デバッグロギングは、デバッグ時にアプリケーションログに詳細情報を追加する方法です。これにより、以下が可能になります。

- アプリケーションの通常の運用時の最小限のロギング出力を維持するため、読みやすさが向上し、ディスク領域の使用量が減少します。
- 問題の解決時にアプリケーションの内部作業に関する詳細情報を表示します。

4.2.1. デバッグロギングの追加

この例では、debug パッケージ を使用しますが、デバッグロギングを処理することのできる 他のパッケージも利用 できます。

前提条件

- デバッグするアプリケーション。例を以下に 示し ます。

手順

1. **デバッグ** ロギング定義を追加します。

```
const debug = require('debug')('myexample');
```

2. デバッグステートメントを追加します。

```
app.use('/api/greeting', (request, response) => {
  const name = request.query ? request.query.name : undefined;
  //log name in debugging
  debug('name: '+name);
  response.send({content: `Hello, ${name || 'World'}`});
});
```

3. デバッグ モジュールを **package.json** に追加します。

```
...
"dependencies": {
    "debug": "^3.1.0"
  }
```

アプリケーションによっては、このモジュールがすでに含まれている可能性があります。たとえば、表現ジェネレーター を使用してアプリケーションを作成すると、**デバッグ** モジュールはすでに **package.json** に追加されます。REST API Level 0 の例などの一部のアプリケーションには、 すでに **package.json** ファイルに **デバッグ** モジュールがあります。

4. アプリケーション依存関係をインストールします。

```
$ npm install
```

4.2.2. localhost のデバッグログへのアクセス

アプリケーションの起動時に **DEBUG** 環境変数を使用してデバッグロギングを有効にします。

前提条件

- デバッグロギングのあるアプリケーション。

手順

1. アプリケーションの起動時に **DEBUG** 環境変数を設定してデバッグロギングを有効にします。

```
$ DEBUG=myexample npm start
```

デバッグ モジュールは ワイルドカード を使用してデバッグメッセージをフィルタリングできます。これは、**DEBUG** 環境変数を使用して設定されます。

2. アプリケーションをテストしてデバッグロギングを呼び出します。
たとえば、REST API Level 0 の例でロギングのデバッグを行う場合は、 **/api/greeting** メソッドで **name** 変数をログに記録するよう設定されます。

```
$ curl http://localhost:8080/api/greeting?name=Sarah
```

3. アプリケーションログを表示し、デバッグメッセージを表示します。

```
myexample name: Sarah +3m
```

4.2.3. OpenShift での Node.js デバッグログへのアクセス

OpenShift のアプリケーション Pod で **DEBUG** 環境変数を使用して、デバッグロギングを有効にします。

前提条件

- デバッグロギングのあるアプリケーション。
- **oc** CLI クライアントがインストールされていること。

手順

1. **oc** CLI クライアントを使用して OpenShift インスタンスにログインします。

```
$ oc login ...
```

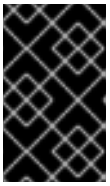
2. アプリケーションを OpenShift にデプロイします。

```
$ npm run openshift
```

これは、ノード shift に直接呼び出しをラップする **openshift** npm スクリプトを実行し ます。

3. Pod の名前を見つけ、ログに従って起動を確認します。

```
$ oc get pods
....
$ oc logs -f pod/POD_NAME
```



重要

Pod の起動後に、このコマンドを実行したままにして、新規ターミナルウインドウで残りの手順を実行します。これにより、ログ **に従い**、作成された新規エントリーを確認できます。

4. アプリケーションをテストします。
たとえば、REST API Level 0 の例にデバッグロギングがあり、 **/api/greeting** メソッドで **name** 変数をログに記録した場合には、以下 を実行します。

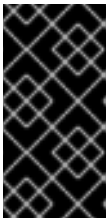
```
$ oc get routes
...
$ curl $APPLICATION_ROUTE/api/greeting?name=Sarah
```

5. Pod ログに戻り、ログにデバッグログメッセージがないことを確認します。
6. **DEBUG** 環境変数を設定してデバッグロギングを有効にします。

```
$ oc get dc
...
$ oc set env dc DC_NAME DEBUG=myexample
```

7. Pod ログに戻り、更新のロールアウトを確認します。
更新がロールアウトされると、Pod は停止し、ログはフォローしなくなります。
8. 新しい Pod の名前を見つけ、ログに従います。

```
$ oc get pods
....
$ oc logs -f pod/POD_NAME
```



重要

Pod の起動後、このコマンドを実行したままにし、別のターミナルウインドウで残りの手順を実行します。これにより、ログ **に従い**、作成された新規エントリーを確認できます。具体的には、ログにはデバッグメッセージが表示されます。

9. アプリケーションをテストしてデバッグロギングを呼び出します。

```
$ oc get routes
...
$ curl $APPLICATION_ROUTE/api/greeting?name=Sarah
```

10. Pod ログに戻り、デバッグメッセージを表示します。

```
...
myexample name: Sarah +3m
```

デバッグロギングを無効にするには、Pod から **DEBUG** 環境変数を削除します。

```
$ oc set env dc DC_NAME DEBUG-
```

関連情報

環境変数の詳細は、OpenShift ドキュメントを参照してください。

第5章 NODE.JS のアプリケーションの例

Node.js ランタイムは、サンプルアプリケーションを提供します。OpenShift でアプリケーションの開発を開始する場合、サンプルアプリケーションをテンプレートとして使用できます。

これらのサンプルアプリケーションは Developer Launcher でアクセスできます。

以下のサンプルアプリケーションをすべてダウンロードしてデプロイできます。

- x86_64 アーキテクチャー： 本ガイドのアプリケーションの例は、x86_64 アーキテクチャーでサンプルアプリケーションをビルドおよびデプロイする方法を説明します。
- s390x アーキテクチャー： IBM Z インフラストラクチャーでプロビジョニングされる OpenShift 環境にサンプルアプリケーションをデプロイするには、コマンドに適切な IBM Z イメージ名を指定します。
  このサンプルアプリケーションでは、ワークフローを実証するために Red Hat Data Grid などの他の製品も必要です。この場合、これらの製品のイメージ名を、サンプルアプリケーションの YAML ファイルの関連する IBM Z イメージ名に変更する必要があります。

注記

Node.js の Secured サンプルアプリケーションには Red Hat SSO 7.3 が必要です。IBM Z では Red Hat SSO 7.3 に対応していないため、IBM Z では Secured の例は利用できません。

5.1. NODE.JS の REST API レベル 0 の例

重要

以下の例は、実稼働環境で実行することは意図されていません。

実験レベルの例： Foundational。

REST API レベル 0 の例の動作

REST API Level 0 の例は、REST フレームワークを使用して HTTP 経由でリモートプロシージャー呼び出しエンドポイントにビジネスオペレーションをマッピングする方法を示しています。これは、Richardson Maturity Model の Level 0 に対応します。REST とその基本原則を使用して HTTP エンドポイントを作成することで、API を柔軟にプロトタイプを作成し、設計することができます。

この例では、HTTP プロトコルを使用してリモートサービスと対話する仕組みを紹介します。これにより、以下が可能になります。

- **api/greeting** エンドポイントで HTTP **GET** リクエストを実行します。
- Hello, World で構成されるペイロードを使用して、JSON 形式で応答を受信します。**文字列。**
- **String 引数を渡す間に api/greeting エンドポイントで HTTP GET リクエストを実行します。これは、クエリー文字列の name request パラメーターを使用します。**
- **$name がリクエストに渡される name パラメーター の値に置き換えられた、ペイロードが**

Hello, $ name! の JSON 形式で応答を受信します。

5.1.1. REST API レベル 0 設計のトレードオフ

表5.1 設計トレードオフ

pros	cons

• サンプルアプリケーションは、迅速にプロトタイピングを有効にします。
• API の設計には柔軟性があります。
• HTTP エンドポイントにより、クライアントは言語に依存しません。
	• アプリケーションまたはサービスの成熟度として、REST API レベル 0 のアプローチは適切にスケーリングされない可能性があります。データベースの対話とクリーンな API 設計またはユースケースをサポートしない場合があります。
	  ◦ 共有可能な、変更可能な状態に関連する操作はすべて、適切なバッキングデータストアと統合する必要があります。
	  ◦ この API 設計で処理されるすべての要求は、要求に対応するコンテナーにのみスコープ指定されます。後続の要求は同じコンテナーによって処理されない可能性があります。

5.1.2. REST API レベル 0 サンプルアプリケーションの OpenShift Online へのデプロイ

OpenShift Online で REST API レベル 0 のサンプルアプリケーションを実行するには、以下のいずれかのオプションを使用します。

- developers.redhat.com/launch の使用
- oc CLI クライアントの使用

各方法は同じ oc コマンドを使用してアプリケーションをデプロイしますが、developers.redhat.com/launch を使用すると、oc コマンドを実行する自動デプロイメントワークフローが提供されます。

5.1.2.1. developers.redhat.com/launch を使用したサンプルアプリケーションのデプロイ

このセクションでは、REST API Level 0 のサンプルアプリケーションをビルドし、Red Hat Developer Launcher Web インターフェースから OpenShift にデプロイする方法を説明します。

前提条件

- OpenShift Online のアカウント。

手順

1. ブラウザーで developers.redhat.com/launch URL に移動します。
2. 画面の指示に従って、Node.js でサンプルアプリケーションを作成して起動します。

5.1.2.2. oc CLI クライアントの認証

oc コマンドラインクライアントを使用して OpenShift Online でアプリケーションのサンプルを使用するには、OpenShift Online Web インターフェースによって提供されるトークンを使用してクライアントを認証する必要があります。

前提条件

- OpenShift Online のアカウント。

手順

1. ブラウザーで OpenShift Online URL に移動します。
2. Web コンソールの右上隅にある疑問符アイコンをクリックします。
3. ドロップダウンメニューで Command Line Tools を選択します。
4. oc login コマンドをコピーします。
5. 端末にコマンドを貼り付けます。このコマンドは、認証トークンを使用して CLI クライアント oc を OpenShift Online アカウントで認証します。

```
$ oc login OPENSHIFT_URL --token=MYTOKEN
```

5.1.2.3. oc CLI クライアントを使用した REST API レベル 0 サンプルアプリケーションのデプロイ

本セクションでは、REST API レベル 0 サンプルアプリケーションをビルドし、コマンドラインから OpenShift にデプロイする方法を説明します。

前提条件

- developers.redhat.com/launch を使用して作成されたサンプルアプリケーション。詳細はを参照してください 「developers.redhat.com/launch を使用したサンプルアプリケーションのデプロイ」。
- 認証された oc クライアント。詳細はを参照してください 「oc CLI クライアントの認証」。

手順

1. GitHub からプロジェクトのクローンを作成します。

```
$ git clone git@github.com:USERNAME/MY_PROJECT_NAME.git
```

プロジェクトの ZIP ファイルをダウンロードした場合は、展開します。

```
$ unzip MY_PROJECT_NAME.zip
```

2. OpenShift で新規プロジェクトを作成します。

```
$ oc new-project MY_PROJECT_NAME
```

3. アプリケーションのルートディレクトリーに移動します。

4. npm を使用して OpenShift へのデプロイメントを開始します。

```
$ npm install && npm run openshift
```

これらのコマンドは、不足しているモジュール依存関係をすべてインストールし、Nodeshift モジュールを使用してサンプルアプリケーションを OpenShift にデプロイしま

す。

5. アプリケーションのステータスを確認し、Pod が実行されていることを確認します。

```
$ oc get pods -w
NAME                        READY     STATUS      RESTARTS   AGE
MY_APP_NAME-1-aaaaa             1/1       Running     0          58s
MY_APP_NAME-s2i-1-build         0/1       Completed   0          2m
```

MY_APP_NAME-1-aaaaa Pod が完全にデプロイされ、起動されると、ステータスが Running である必要があります。特定の Pod 名が異なります。新規ビルドごとに、中程度の数字が増加します。末尾の文字は、Pod の作成時に生成されます。

6. サンプルアプリケーションがデプロイされ、起動したら、そのルートを決定します。

ルート情報の例

```
$ oc get routes
NAME              HOST/PORT                                        PATH      SERVICES
PORT      TERMINATION
MY_APP_NAME       MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME
MY_APP_NAME      8080
```

Pod のルート情報は、アクセスに使用するベース URL を提供します。上記の例では、アプリケーションにアクセス する ためにベース URL として http://MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME を使用します。

5.1.3. REST API レベル 0 サンプルアプリケーションの Minishift または CDK へのデプロイ

以下のいずれかのオプションを使用して、Minishift または CDK で REST API レベル 0 のサンプルアプリケーションをローカルに実行します。

- Fabric8 Launcher の使用

- oc CLI クライアントの使用

各方法は同じ oc コマンドを使用してアプリケーションをデプロイしますが、Fabric8 Launcher を使用すると oc コマンドを実行する自動デプロイメントワークフローが提供されます。

5.1.3.1. Fabric8 Launcher ツールの URL および認証情報の取得

Minishift または CDK でサンプルアプリケーションを作成してデプロイするには、Fabric8 Launcher ツール URL およびユーザー認証情報が必要です。この情報は、Minishift または CDK の起動時に提供されます。

前提条件

- Fabric8 Launcher ツールがインストールされ、設定されている。

手順

1. Minishift または CDK を起動したコンソールに移動します。

2. 実行中の Fabric8 Launcher にアクセスするために使用できる URL およびユーザー認証情報のコンソール出力を確認します。

Minishift または CDK 起動からのコンソール出力の例

```
...
-- Removing temporary directory ... OK
-- Server Information ...
   OpenShift server started.
   The server is accessible via web console at:
      https://192.168.42.152:8443

   You are logged in as:
      User:     developer
      Password: developer

   To login as administrator:
      oc login -u system:admin
```

5.1.3.2. Fabric8 Launcher ツールを使用したサンプルアプリケーションのデプロイ

本セクションでは、REST API レベル 0 サンプルアプリケーションをビルドし、それを Fabric8 Launcher Web インターフェースから OpenShift にデプロイする方法を説明します。

前提条件

- 実行中の Fabric8 Launcher インスタンスの URL と Minishift または CDK のユーザー認証情報。詳細はを参照してください 「Fabric8 Launcher ツールの URL および認証情報の取得」。

手順

1. ブラウザーで Fabric8 Launcher URL に移動します。
2. 画面の指示に従って、Node.js でサンプルアプリケーションを作成して起動します。

5.1.3.3. oc CLI クライアントの認証

oc コマンドラインクライアントを使用して Minishift または CDK でアプリケーションの例を使用するには、Minishift または CDK Web インターフェースが提供するトークンを使用してクライアントを認証する必要があります。

前提条件

- 実行中の Fabric8 Launcher インスタンスの URL と Minishift または CDK のユーザー認証情報。詳細はを参照してください 「Fabric8 Launcher ツールの URL および認証情報の取得」。

手順

1. ブラウザーで Minishift または CDK URL に移動します。
2. Web コンソールの右上隅にある疑問符アイコンをクリックします。

3. ドロップダウンメニューで Command Line Tools を選択します。

4. oc login コマンドをコピーします。

5. 端末にコマンドを貼り付けます。このコマンドは、認証トークンを使用して Minishift または CDK アカウントで oc CLI クライアントを認証します。

```
$ oc login OPENSHIFT_URL --token=MYTOKEN
```

5.1.3.4. oc CLI クライアントを使用した REST API レベル 0 サンプルアプリケーションのデプロイ

本セクションでは、REST API レベル 0 サンプルアプリケーションをビルドし、コマンドラインから OpenShift にデプロイする方法を説明します。

前提条件

- Minishift または CDK で Fabric8 Launcher ツールを使用して作成されたアプリケーションのサンプル。詳細はを参照してください 「Fabric8 Launcher ツールを使用したサンプルアプリケーションのデプロイ」。
- Fabric8 Launcher ツール URL。
- 認証された oc クライアント。詳細はを参照してください 「oc CLI クライアントの認証」。

手順

1. GitHub からプロジェクトのクローンを作成します。

```
$ git clone git@github.com:USERNAME/MY_PROJECT_NAME.git
```

プロジェクトの ZIP ファイルをダウンロードした場合は、展開します。

```
$ unzip MY_PROJECT_NAME.zip
```

2. OpenShift で新規プロジェクトを作成します。

```
$ oc new-project MY_PROJECT_NAME
```

3. アプリケーションのルートディレクトリーに移動します。

4. npm を使用して OpenShift へのデプロイメントを開始します。

```
$ npm install && npm run openshift
```

これらのコマンドは、不足しているモジュール依存関係をすべてインストールし、Nodeshift モジュールを使用してサンプルアプリケーションを OpenShift にデプロイします。

5. アプリケーションのステータスを確認し、Pod が実行されていることを確認します。

```
$ oc get pods -w
NAME                     READY     STATUS      RESTARTS   AGE
MY_APP_NAME-1-aaaaa            1/1       Running     0          58s
MY_APP_NAME-s2i-1-build        0/1       Completed   0          2m
```

MY_APP_NAME-1-aaaaa Pod が完全にデプロイされ、起動されると、ステータスが Running である必要があります。特定の Pod 名が異なります。新規ビルドごとに、中程度の数字が増加します。末尾の文字は、Pod の作成時に生成されます。

6. サンプルアプリケーションがデプロイされ、起動したら、そのルートを決定します。

ルート情報の例

```
$ oc get routes
NAME              HOST/PORT                                        PATH      SERVICES
PORT      TERMINATION
MY_APP_NAME       MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME
MY_APP_NAME      8080
```

Pod のルート情報は、アクセスに使用するベース URL を提供します。上記の例では、アプリケーションにアクセス する ためにベース URL として http://MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME を使用します。

5.1.4. REST API レベル 0 サンプルアプリケーションの OpenShift Container Platform へのデプロイ

サンプルアプリケーションを OpenShift Container Platform に作成し、デプロイするプロセスは OpenShift Online と似ています。

前提条件

- developers.redhat.com/launch を使用して作成されたサンプルアプリケーション。

手順

- の手順に従って 「REST API レベル 0 サンプルアプリケーションの OpenShift Online へのデプロイ」、OpenShift Container Platform Web コンソールからの URL およびユーザー認証情報のみを使用します。

5.1.5. Node.js 用の変更されていない REST API レベル 0 のサンプルアプリケーションとの対話

この例では、GET リクエストを許可するデフォルトの HTTP エンドポイントを提供します。

前提条件

- アプリケーションを実行している。
- curl バイナリーまたは Web ブラウザー

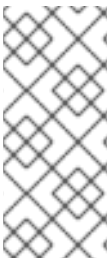
手順

1. curl を使用して、例に対して GET 要求を実行します。ブラウザーを使用してこれを行うこともできます。

```
$ curl http://MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME/api/greeting
{"content":"Hello, World!"}
```

2. curl を使用して、例に対して URL パラメーター で GET リクエストを実行します。ブラウザーを使用してこれを行うこともできます。

```
$ curl http://MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME/api/greeting?name=Sarah
{"content":"Hello, Sarah!"}
```



注記

ブラウザーから、この例で提供されるフォームを使用して、同じ対話を実行することもできます。このフォームは、プロジェクト http://MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME のルートにあり ます。

5.1.6. REST リソース

REST に関する背景や関連情報は、以下を参照してください。

- アーキテクチャーおよびネットワークベースのソフトウェアアーキテクチャーの設計 - Representational State Transfer(REST)
- parkardson Maturity Model
- 表現 Web Framework
- Spring Boot の REST API レベル 0
- Eclipse Vert.x の REST API レベル 0
- Thorntail 用の REST API レベル 0

5.2. NODE.JS の外部化設定例

重要

以下の例は、実稼働環境で実行することは意図されていません。

実験レベルの例： Foundational。

外部化設定では、ConfigMap を使用した外部化設定の基本的な例が提供されます。ConfigMap は、コンテナーを OpenShift から独立させ、1 つ以上の Linux コンテナーに単純なキーと値のペアとして設定データをインジェクトするために OpenShift によって使用されるオブジェクトです。

この例では、以下の方法を紹介します。

- ConfigMap をセットアップし、設定します。
- アプリケーション内の ConfigMap が提供する設定を使用します。
- 実行中のアプリケーションの ConfigMap 設定への変更をデプロイします。

5.2.1. 外部化設定設計パターン

可能な場合は、アプリケーション設定を外部化して、アプリケーションコードから分離します。これにより、異なる環境を通過する際にアプリケーションの設定を変更できますが、コードは変更されません。また、設定を外部化してコードベースおよびバージョン管理から機密情報や内部情報を保持します。多くの言語およびアプリケーションサーバーは、アプリケーション設定の外部化をサポートする環境変数を提供します。

マイクロサービスアーキテクチャーとマルチ言語（ポルトガル語）環境は、アプリケーションの設定を管理する際の複雑性の層を追加します。アプリケーションは独立した分散サービスで構成されており、それぞれに独自の設定を指定できます。すべての設定データを同期してアクセス可能なと、メンテナンスの課題が発生する可能性があります。

ConfigMap により、アプリケーション設定を外部化し、OpenShift の個別の Linux コンテナーおよび Pod で使用できるようになります。YAML ファイルを使用して、これを Linux コンテナーに挿入するなど、さまざまな方法で ConfigMap オブジェクトを作成できます。ConfigMap により、設定データのグループ化およびスケーリングも可能です。これにより、基本的な 開発、ステージ、および 実稼働

環境以外に、多くの環境を設定できます。ConfigMap の詳細は、OpenShift ドキュメント を参照してください。

5.2.2. 外部化設定設計のトレードオフ

表5.2 Design Tradeoffs

pros	cons
● 設定はデプロイメントとは別のものです ● 個別に更新可能 ● サービス間で共有が可能	● 環境への設定を追加するには、追加手順が必要です。 ● 別々に維持する必要があります ● サービスの範囲外の調整が必要

5.2.3. 外部化設定サンプルアプリケーションの OpenShift Online へのデプロイ

OpenShift Online で Externalized Configuration サンプルアプリケーションを実行するには、以下のいずれかのオプションを使用します。

- developers.redhat.com/launch の使用
- oc CLI クライアントの使用

各方法は同じ oc コマンドを使用してアプリケーションをデプロイしますが、developers.redhat.com/launch を使用すると、oc コマンドを実行する自動デプロイメントワークフローが提供されます。

5.2.3.1. developers.redhat.com/launch を使用したサンプルアプリケーションのデプロイ

このセクションでは、REST API Level 0 のサンプルアプリケーションをビルドし、Red Hat Developer Launcher Web インターフェースから OpenShift にデプロイする方法を説明します。

前提条件

- OpenShift Online のアカウント。

手順

1. ブラウザーで developers.redhat.com/launch URL に移動します。

2. 画面の指示に従って、Node.js でサンプルアプリケーションを作成して起動します。

5.2.3.2. oc CLI クライアントの認証

oc コマンドラインクライアントを使用して OpenShift Online でアプリケーションのサンプルを使用するには、OpenShift Online Web インターフェースによって提供されるトークンを使用してクライアントを認証する必要があります。

前提条件

- OpenShift Online のアカウント。

手順

1. ブラウザーで OpenShift Online URL に移動します。

2. Web コンソールの右上隅にある疑問符アイコンをクリックします。

3. ドロップダウンメニューで Command Line Tools を選択します。

4. oc login コマンドをコピーします。

5. 端末にコマンドを貼り付けます。このコマンドは、認証トークンを使用して CLI クライアント oc を OpenShift Online アカウントで認証します。

```
$ oc login OPENSHIFT_URL --token=MYTOKEN
```

5.2.3.3. oc CLI クライアントを使用した外部化設定サンプルアプリケーションのデプロイ

このセクションでは、コマンドラインから外部化設定サンプルアプリケーションをビルドし、これを OpenShift にデプロイする方法を説明します。

前提条件

- developers.redhat.com/launch を使用して作成されたサンプルアプリケーション。詳細はを参照してください 「developers.redhat.com/launch を使用したサンプルアプリケーションのデプロイ」。

- 認証された oc クライアント。詳細はを参照してください 「oc CLI クライアントの認証」。

手順

1. GitHub からプロジェクトのクローンを作成します。

```
$ git clone git@github.com:USERNAME/MY_PROJECT_NAME.git
```

プロジェクトの ZIP ファイルをダウンロードした場合は、展開します。

```
$ unzip MY_PROJECT_NAME.zip
```

2. 新しい OpenShift プロジェクトを作成します。

```
$ oc new-project MY_PROJECT_NAME
```

3. サンプルアプリケーションをデプロイする前にサービスアカウントに表示アクセス権限を割り当て、アプリケーションが ConfigMap の内容を読み取るために OpenShift API にアクセスできるようにします。

```
$ oc policy add-role-to-user view -n $(oc project -q) -z default
```

4. アプリケーションのルートディレクトリーに移動します。

5. app-config.yml を使用して ConfigMap 設定を OpenShift にデプロイします。

```
$ oc create configmap app-config --from-file=app-config.yml
```

6. ConfigMap 設定がデプロイされていることを確認します。

```
$ oc get configmap app-config -o yaml

apiVersion: template.openshift.io/v1
data:
  app-config.yml: |-
    message : "Hello, %s from a ConfigMap !"
    level : INFO
...
```

7. npm を使用して OpenShift へのデプロイメントを開始します。

```
$ npm install && npm run openshift
```

これらのコマンドは、不足しているモジュール依存関係をすべてインストールし、Nodeshift モジュールを使用してサンプルアプリケーションを OpenShift にデプロイします。

8. アプリケーションのステータスを確認し、Pod が実行されていることを確認します。

```
$ oc get pods -w
NAME                          READY    STATUS     RESTARTS   AGE
MY_APP_NAME-1-aaaaa           1/1      Running    0          58s
MY_APP_NAME-s2i-1-build       0/1      Completed  0          2m
```

MY_APP_NAME-1-aaaaa Pod の完全なデプロイおよび起動後に、ステータスが Running である必要があります。特定の Pod 名が異なります。新規ビルドごとに、中程度の数字が増加します。末尾の文字は、Pod の作成時に生成されます。

9. サンプルアプリケーションがデプロイされ、起動したら、そのルートを決定します。

ルート情報の例

```
$ oc get routes
NAME              HOST/PORT                                        PATH      SERVICES
PORT      TERMINATION
MY_APP_NAME       MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME
MY_APP_NAME      8080
```

Pod のルート情報は、アクセスに使用するベース URL を提供します。上記の例では、アプリケーションにアクセス する ためにベース URL として http://MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME を使用します。

5.2.4. Minishift または CDK への Externalized Configuration サンプルアプリケーションのデプロイ

以下のいずれかのオプションを使用して、Minishift または CDK で Externalized Configuration サンプルアプリケーションをローカルに実行します。

- Fabric8 Launcher の使用
- oc CLI クライアントの使用

各方法は同じ oc コマンドを使用してアプリケーションをデプロイしますが、Fabric8 Launcher を使用すると oc コマンドを実行する自動デプロイメントワークフローが提供されます。

5.2.4.1. Fabric8 Launcher ツールの URL および認証情報の取得

Minishift または CDK でサンプルアプリケーションを作成してデプロイするには、Fabric8 Launcher ツール URL およびユーザー認証情報が必要です。この情報は、Minishift または CDK の起動時に提供されます。

前提条件

- Fabric8 Launcher ツールがインストールされ、設定されている。

手順

1. Minishift または CDK を起動したコンソールに移動します。
2. 実行中の Fabric8 Launcher にアクセスするために使用できる URL およびユーザー認証情報のコンソール出力を確認します。

Minishift または CDK 起動からのコンソール出力の例

-

```
...
-- Removing temporary directory ... OK
-- Server Information ...
   OpenShift server started.
   The server is accessible via web console at:
      https://192.168.42.152:8443

   You are logged in as:
      User:     developer
      Password: developer

   To login as administrator:
      oc login -u system:admin
```

5.2.4.2. Fabric8 Launcher ツールを使用したサンプルアプリケーションのデプロイ

本セクションでは、REST API レベル 0 サンプルアプリケーションをビルドし、それを Fabric8 Launcher Web インターフェースから OpenShift にデプロイする方法を説明します。

前提条件

- 実行中の Fabric8 Launcher インスタンスの URL と Minishift または CDK のユーザー認証情報。詳細はを参照してください 「Fabric8 Launcher ツールの URL および認証情報の取得」。

手順

1. ブラウザーで Fabric8 Launcher URL に移動します。

2. 画面の指示に従って、Node.js でサンプルアプリケーションを作成して起動します。

5.2.4.3. oc CLI クライアントの認証

oc コマンドラインクライアントを使用して Minishift または CDK でアプリケーションの例を使用するには、Minishift または CDK Web インターフェースが提供するトークンを使用してクライアントを認証する必要があります。

前提条件

-

実行中の Fabric8 Launcher インスタンスの URL と Minishift または CDK のユーザー認証情報。詳細はを参照してください 「Fabric8 Launcher ツールの URL および認証情報の取得」。

手順

1. ブラウザーで Minishift または CDK URL に移動します。
2. Web コンソールの右上隅にある疑問符アイコンをクリックします。
3. ドロップダウンメニューで Command Line Tools を選択します。
4. oc login コマンドをコピーします。
5. 端末にコマンドを貼り付けます。このコマンドは、認証トークンを使用して Minishift または CDK アカウントで oc CLI クライアントを認証します。

```
$ oc login OPENSHIFT_URL --token=MYTOKEN
```

5.2.4.4. oc CLI クライアントを使用した外部化設定サンプルアプリケーションのデプロイ

このセクションでは、コマンドラインから外部化設定サンプルアプリケーションをビルドし、これを OpenShift にデプロイする方法を説明します。

前提条件

- Minishift または CDK で Fabric8 Launcher ツールを使用して作成されたアプリケーションのサンプル。詳細はを参照してください 「Fabric8 Launcher ツールを使用したサンプルアプリケーションのデプロイ」。
- Fabric8 Launcher ツール URL。
- 認証された oc クライアント。詳細はを参照してください 「oc CLI クライアントの認証」。

手順

1. GitHub からプロジェクトのクローンを作成します。

```
$ git clone git@github.com:USERNAME/MY_PROJECT_NAME.git
```

プロジェクトの ZIP ファイルをダウンロードした場合は、展開します。

```
$ unzip MY_PROJECT_NAME.zip
```

2. 新しい OpenShift プロジェクトを作成します。

```
$ oc new-project MY_PROJECT_NAME
```

3. サンプルアプリケーションをデプロイする前にサービスアカウントに表示アクセス権限を割り当て、アプリケーションが ConfigMap の内容を読み取るために OpenShift API にアクセスできるようにします。

```
$ oc policy add-role-to-user view -n $(oc project -q) -z default
```

4. アプリケーションのルートディレクトリーに移動します。

5. app-config.yml を使用して ConfigMap 設定を OpenShift にデプロイします。

```
$ oc create configmap app-config --from-file=app-config.yml
```

6. ConfigMap 設定がデプロイされていることを確認します。

```
$ oc get configmap app-config -o yaml

apiVersion: template.openshift.io/v1
data:
  app-config.yml: |-
    message : "Hello, %s from a ConfigMap !"
    level : INFO
...
```

7. npm を使用して OpenShift へのデプロイメントを開始します。

```
$ npm install && npm run openshift
```

これらのコマンドは、不足しているモジュール依存関係をすべてインストールし、Nodeshift モジュールを使用してサンプルアプリケーションを OpenShift にデプロイします。

8. アプリケーションのステータスを確認し、Pod が実行されていることを確認します。

```
$ oc get pods -w
NAME                              READY     STATUS      RESTARTS   AGE
MY_APP_NAME-1-aaaaa               1/1       Running     0          58s
MY_APP_NAME-s2i-1-build           0/1       Completed   0          2m
```

MY_APP_NAME-1-aaaaa Pod の完全なデプロイおよび起動後に、ステータスが Running である必要があります。特定の Pod 名が異なります。新規ビルドごとに、中程度の数字が増加します。末尾の文字は、Pod の作成時に生成されます。

9. サンプルアプリケーションがデプロイされ、起動したら、そのルートを決定します。

ルート情報の例

```
$ oc get routes
NAME              HOST/PORT                                        PATH      SERVICES
PORT      TERMINATION
MY_APP_NAME       MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME
MY_APP_NAME      8080
```

Pod のルート情報は、アクセスに使用するベース URL を提供します。上記の例では、アプリケーションにアクセス する ためにベース URL として http://MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME を使用します。

5.2.5. 外部化設定サンプルアプリケーションの OpenShift Container Platform へのデプロイ

サンプルアプリケーションを OpenShift Container Platform に作成し、デプロイするプロセスは OpenShift Online と似ています。

前提条件

- developers.redhat.com/launch を使用して作成されたサンプルアプリケーション。

手順

- の手順に従って「外部化設定サンプルアプリケーションの OpenShift Online へのデプロイ」、OpenShift Container Platform Web コンソールからの URL およびユーザー認証情報のみを使用します。

5.2.6. Node.js の未変更の外部化設定例のアプリケーションとの対話

この例は、GET リクエストを受け入れるデフォルトの HTTP エンドポイントを提供します。

前提条件

- アプリケーションを実行している。
- curl バイナリーまたは Web ブラウザー

手順

1. curl を使用して、例に対して GET 要求を実行します。ブラウザーを使用してこれを行うこともできます。

```
$ curl http://MY_APP_NAME-
MY_PROJECT_NAME.OPENSHIFT_HOSTNAME/api/greeting
{"content":"Hello, World from a ConfigMap !"}
```

2. デプロイされた ConfigMap 設定を更新します。

```
$ oc edit configmap app-config
```

メッセージ キーの値を Bonjour、%s から ConfigMap ! に変更し、ファイルを保存します。

3. ConfigMap の更新は、アプリケーションの再起動を必要とせずに、許容期間（数秒）内でアプリケーションによって読み取る必要があります。

4. 更新された ConfigMap 設定の例で curl を使用して GET リクエストを実行し、更新された greeting を確認します。また、アプリケーションが提供する Web フォームを使用して、ブラウザーからこの作業を行うこともできます。

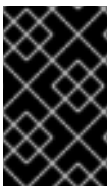
```
$ curl http://MY_APP_NAME-
MY_PROJECT_NAME.OPENSHIFT_HOSTNAME/api/greeting
{"content":"Bonjour, World from a ConfigMap !"}
```

5.2.7. 外部化設定リソース

外部化設定および ConfigMap の背景および関連情報は、以下を参照してください。

- OpenShift ConfigMap ドキュメント
- OpenShift の ConfigMap について
- Spring Boot の外部化設定
- Eclipse Vert.x の外部化設定
- Thorntail の外部化設定

5.3. NODE.JS のリレーショナルデータベースバックエンドの例



重要

以下の例は、実稼働環境で実行することは意図されていません。

制限： Minishift または CDK でこのサンプルアプリケーションを実行します。手動でのワークフローを使用して、この例を OpenShift Online Pro および OpenShift Container Platform にデプロイすることもできます。この例では、現在 OpenShift Online ドキュメンテーションでは使用できません。

実験レベルの例： Foundational。

リレーショナルデータベースバックエンドの例

Relational Database バックエンドの例は、REST API Level 0 アプリケーションで拡張し、単純な HTTP API を使用して PostgreSQL データベースで 作成、読み取り、更新、削除 (CRUD)操作を実行する基本的な例を提供します。CRUD 操作は永続ストレージの基本機能で、HTTP API がデータベースを処理する際に広く使用されている機能です。

この例では、HTTP アプリケーションの OpenShift でデータベースを検索し、接続する機能も例示します。各ランタイムは、指定のケースで最適な接続ソリューションを実装する方法を示します。ランタイムは、JDBC、JPA、または ORM API を直接アクセスするなど、オプションを選択できます。

サンプルアプリケーションは HTTP API を公開し、HTTP で CRUD 操作を実行してデータを操作できるエンドポイントを提供します。CRUD 操作は HTTP Verbs にマップされます。API は JSON 形式でリクエストを受信し、ユーザーに応答を返します。ユーザーは、例で提供されるユーザーインターフェースを使用してアプリケーションを使用できます。具体的には、以下の例で以下を可能にするアプリケーションを提供します。

- ブラウザーでアプリケーション Web インターフェースに移動します。これにより、my_data データベースのデータで CRUD 操作を実行できる単純な Web サイトが提供されます。
- api/fruits エンドポイントで HTTP GET 要求を実行します。
- データベースにすべての fruits 一覧が含まれる JSON 配列としてフォーマットされた応答を受け取ります。
- 有効なアイテム ID を引数として渡す間に、api/fruits/* エンドポイントで HTTP GET リクエストを実行します。
- 指定の ID の fruit の名前が含まれる JSON 形式で応答を受け取ります。指定された ID と一致する項目がない場合、呼び出しは HTTP エラー 404 になります。
- 有効な 名前 値を渡す api/fruits エンドポイントで HTTP POST 要求を実行し、データベースに新規エントリーを作成します。

- 有効な ID および名前を引数として渡す api/fruits/* エンドポイントで HTTP PUT 要求を実行します。これにより、要求で指定した名前に一致するように、指定の ID を持つ項目の名前が更新されます。

- api/fruits/* エンドポイントで HTTP DELETE 要求を実行し、有効な ID を引数として渡します。これにより、データベースから指定の ID のアイテムが削除され、レスポンスとして HTTP コード 204 (No Content)を返します。無効な ID を渡すと、呼び出しは HTTP エラー 404 となります。

この例では、完全な成熟した RESTful モデル（レベル 3）は表示されませんが、推奨される HTTP API プラクティスに従って、互換性のある HTTP 動詞およびステータスを使用します。

5.3.1. リレーショナルデータベースバックエンド設計のトレードオフ

表5.3 Design Tradeoffs

pros	cons
● 各ランタイムは、データベースの対話の実装方法を決定します。JDBC などの低レベルの接続 API も、他の JPA も使用でき、もう 1 つは ORM API に直接アクセスできます。各ランタイムは最適な方法を決定します。 ● 各ランタイムはスキーマの作成方法を決定します。	● このサンプルアプリケーションで提供される PostgreSQL データベースは、永続ストレージでバックアップされません。データベース Pod を停止または再デプロイすると、データベースへの変更は失われます。変更を保持するために、サンプルアプリケーションの Pod で外部データベースを使用するには、OpenShift ドキュメントの「Creating an application with a database」を参照してください。OpenShift でデータベースコンテナーを使用して永続ストレージを設定することもできます。OpenShift およびコンテナーで永続ストレージを使用する方法は、OpenShift ドキュメントの「Persistent Storage」、「Managing Volumes」および「Persistent Volumes」の章を参照してください。

5.3.2. OpenShift Online への Relational Database バックエンドサンプルアプリケーションのデプロイ

以下のオプションのいずれかを使用して、OpenShift Online で Relational Database バックエンドサンプルアプリケーションを実行します。

- developers.redhat.com/launch の使用

- oc CLI クライアントの使用

各方法は同じ oc コマンドを使用してアプリケーションをデプロイしますが、developers.redhat.com/launch を使用すると、oc コマンドを実行する自動デプロイメントワークフローが提供されます。

5.3.2.1. developers.redhat.com/launch を使用したサンプルアプリケーションのデプロイ

このセクションでは、REST API Level 0 のサンプルアプリケーションをビルドし、Red Hat Developer Launcher Web インターフェースから OpenShift にデプロイする方法を説明します。

前提条件

- OpenShift Online のアカウント。

手順

1. ブラウザーで developers.redhat.com/launch URL に移動します。
2. 画面の指示に従って、Node.js でサンプルアプリケーションを作成して起動します。

5.3.2.2. oc CLI クライアントの認証

oc コマンドラインクライアントを使用して OpenShift Online でアプリケーションのサンプルを使用するには、OpenShift Online Web インターフェースによって提供されるトークンを使用してクライアントを認証する必要があります。

前提条件

- OpenShift Online のアカウント。

手順

1. ブラウザーで OpenShift Online URL に移動します。

2. Web コンソールの右上隅にある疑問符アイコンをクリックします。

3. ドロップダウンメニューで Command Line Tools を選択します。

4. oc login コマンドをコピーします。

5. 端末にコマンドを貼り付けます。このコマンドは、認証トークンを使用して CLI クライアント oc を OpenShift Online アカウントで認証します。

```
$ oc login OPENSHIFT_URL --token=MYTOKEN
```

5.3.2.3. oc CLI クライアントを使用した Relational Database バックエンドサンプルアプリケーションのデプロイ

本セクションでは、Relational Database バックエンドのサンプルアプリケーションをビルドし、コマンドラインから OpenShift にデプロイする方法を説明します。

前提条件

- developers.redhat.com/launch を使用して作成されたサンプルアプリケーション。詳細はを参照してください 「developers.redhat.com/launch を使用したサンプルアプリケーションのデプロイ」。
- 認証された oc クライアント。詳細はを参照してください 「oc CLI クライアントの認証」。

手順

1. GitHub からプロジェクトのクローンを作成します。

```
$ git clone git@github.com:USERNAME/MY_PROJECT_NAME.git
```

プロジェクトの ZIP ファイルをダウンロードした場合は、展開します。

```
$ unzip MY_PROJECT_NAME.zip
```

2. 新しい OpenShift プロジェクトを作成します。

```
$ oc new-project MY_PROJECT_NAME
```

3. アプリケーションのルートディレクトリーに移動します。

4. PostgreSQL データベースを OpenShift にデプロイします。データベースアプリケーションの作成時に、ユーザー名、パスワード、およびデータベース名に以下の値を使用するようにしてください。サンプルアプリケーションは、これらの値を使用するように事前に設定されています。異なる値を使用すると、アプリケーションによるデータベースの統合ができなくなります。

```
$ oc new-app -e POSTGRESQL_USER=luke -ePOSTGRESQL_PASSWORD=secret -
ePOSTGRESQL_DATABASE=my_data registry.access.redhat.com/rhscl/postgresql-
10-rhel7 --name=my-database
```

5. データベースのステータスを確認し、Pod が実行されていることを確認します。

```
$ oc get pods -w
my-database-1-aaaaa   1/1       Running   0         45s
my-database-1-deploy   0/1       Completed   0         53s
```

my-database-1-aaaaa Pod のステータスは Running であり、完全にデプロイされ、起動されると ready と表示されるはずです。特定の Pod 名が異なります。新規ビルドごとに、中程度の数字が増加します。末尾の文字は、Pod の作成時に生成されます。

6. npm を使用して OpenShift へのデプロイメントを開始します。

```
$ npm install && npm run openshift
```

これらのコマンドは、不足しているモジュール依存関係をすべてインストールし、Nodeshift モジュールを使用してサンプルアプリケーションを OpenShift にデプロイします。

7. アプリケーションのステータスを確認し、Pod が実行されていることを確認します。

```
$ oc get pods -w
NAME                      READY     STATUS      RESTARTS   AGE
MY_APP_NAME-1-aaaaa       1/1       Running     0          58s
```

```
MY_APP_NAME-s2i-1-build   0/1       Completed   0         2m
```

MY_APP_NAME-1-aaaaa Pod のステータスは Running であり、完全にデプロイされ、起動する準備が整った時点で示される必要があります。

8. サンプルアプリケーションがデプロイされ、起動したら、そのルートを決定します。

ルート情報の例

```
$ oc get routes
NAME            HOST/PORT                                PATH      SERVICES          PORT
TERMINATION
MY_APP_NAME   MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME
MY_APP_NAME   8080
```

Pod のルート情報は、アクセスに使用するベース URL を提供します。上記の例では、アプリケーションにアクセス する ためにベース URL として http://MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME を使用します。

5.3.3. Minishift または CDK への Relational Database バックエンドサンプルアプリケーションのデプロイ

以下のいずれかのオプションを使用して、Minishift または CDK で Relational Database バックエンドサンプルアプリケーションをローカルに実行します。

- Fabric8 Launcher の使用
- oc CLI クライアントの使用

各方法は同じ oc コマンドを使用してアプリケーションをデプロイしますが、Fabric8 Launcher を使用すると oc コマンドを実行する自動デプロイメントワークフローが提供されます。

5.3.3.1. Fabric8 Launcher ツールの URL および認証情報の取得

Minishift または CDK でサンプルアプリケーションを作成してデプロイするには、Fabric8 Launcher ツール URL およびユーザー認証情報が必要です。この情報は、Minishift または CDK の起動時に提供されます。

前提条件

- Fabric8 Launcher ツールがインストールされ、設定されている。

手順

1. Minishift または CDK を起動したコンソールに移動します。

2. 実行中の Fabric8 Launcher にアクセスするために使用できる URL およびユーザー認証情報のコンソール出力を確認します。

Minishift または CDK 起動からのコンソール出力の例

```
...
-- Removing temporary directory ... OK
-- Server Information ...
   OpenShift server started.
   The server is accessible via web console at:
      https://192.168.42.152:8443

   You are logged in as:
      User:     developer
      Password: developer

   To login as administrator:
      oc login -u system:admin
```

5.3.3.2. Fabric8 Launcher ツールを使用したサンプルアプリケーションのデプロイ

本セクションでは、REST API レベル 0 サンプルアプリケーションをビルドし、それを Fabric8 Launcher Web インターフェースから OpenShift にデプロイする方法を説明します。

前提条件

- 実行中の Fabric8 Launcher インスタンスの URL と Minishift または CDK のユーザー認証情報。詳細はを参照してください 「Fabric8 Launcher ツールの URL および認証情報の取得」。

手順

1. ブラウザーで Fabric8 Launcher URL に移動します。

2. 画面の指示に従って、Node.js でサンプルアプリケーションを作成して起動します。

5.3.3.3. oc CLI クライアントの認証

oc コマンドラインクライアントを使用して Minishift または CDK でアプリケーションの例を使用するには、Minishift または CDK Web インターフェースが提供するトークンを使用してクライアントを認証する必要があります。

前提条件

- 実行中の Fabric8 Launcher インスタンスの URL と Minishift または CDK のユーザー認証情報。詳細はを参照してください 「Fabric8 Launcher ツールの URL および認証情報の取得」。

手順

1. ブラウザーで Minishift または CDK URL に移動します。

2. Web コンソールの右上隅にある疑問符アイコンをクリックします。

3. ドロップダウンメニューで Command Line Tools を選択します。

4. oc login コマンドをコピーします。

5. 端末にコマンドを貼り付けます。このコマンドは、認証トークンを使用して Minishift または CDK アカウントで oc CLI クライアントを認証します。

```
$ oc login OPENSHIFT_URL --token=MYTOKEN
```

5.3.3.4. oc CLI クライアントを使用した Relational Database バックエンドサンプルアプリケーションのデプロイ

本セクションでは、Relational Database バックエンドのサンプルアプリケーションをビルドし、コマンドラインから OpenShift にデプロイする方法を説明します。

前提条件

- Minishift または CDK で Fabric8 Launcher ツールを使用して作成されたアプリケーションのサンプル。詳細はを参照してください 「Fabric8 Launcher ツールを使用したサンプルアプリケーションのデプロイ」。
- Fabric8 Launcher ツール URL。
- 認証された oc クライアント。詳細はを参照してください 「oc CLI クライアントの認証」。

手順

1. GitHub からプロジェクトのクローンを作成します。

```
$ git clone git@github.com:USERNAME/MY_PROJECT_NAME.git
```

プロジェクトの ZIP ファイルをダウンロードした場合は、展開します。

```
$ unzip MY_PROJECT_NAME.zip
```

2. 新しい OpenShift プロジェクトを作成します。

```
$ oc new-project MY_PROJECT_NAME
```

3. アプリケーションのルートディレクトリーに移動します。

4. PostgreSQL データベースを OpenShift にデプロイします。データベースアプリケーショ

ンの作成時に、ユーザー名、パスワード、およびデータベース名に以下の値を使用するようにしてください。サンプルアプリケーションは、これらの値を使用するように事前に設定されています。異なる値を使用すると、アプリケーションによるデータベースの統合ができなくなります。

```
$ oc new-app -e POSTGRESQL_USER=luke -ePOSTGRESQL_PASSWORD=secret -
ePOSTGRESQL_DATABASE=my_data registry.access.redhat.com/rhscl/postgresql-
10-rhel7 --name=my-database
```

5. データベースのステータスを確認し、Pod が実行されていることを確認します。

```
$ oc get pods -w
my-database-1-aaaaa   1/1       Running   0         45s
my-database-1-deploy   0/1       Completed   0         53s
```

my-database-1-aaaaa Pod のステータスは Running であり、完全にデプロイされ、起動されると ready と表示されるはずです。特定の Pod 名が異なります。新規ビルドごとに、中程度の数字が増加します。末尾の文字は、Pod の作成時に生成されます。

6. npm を使用して OpenShift へのデプロイメントを開始します。

```
$ npm install && npm run openshift
```

これらのコマンドは、不足しているモジュール依存関係をすべてインストールし、Nodeshift モジュールを使用してサンプルアプリケーションを OpenShift にデプロイします。

7. アプリケーションのステータスを確認し、Pod が実行されていることを確認します。

```
$ oc get pods -w
NAME                      READY     STATUS      RESTARTS   AGE
MY_APP_NAME-1-aaaaa       1/1       Running     0          58s
MY_APP_NAME-s2i-1-build   0/1       Completed   0          2m
```

MY_APP_NAME-1-aaaaa Pod のステータスは Running であり、完全にデプロイされ、起動する準備が整った時点で示される必要があります。

8. サンプルアプリケーションがデプロイされ、起動したら、そのルートを決定します。

ルート情報の例

```
$ oc get routes
NAME            HOST/PORT                                   PATH      SERVICES          PORT
TERMINATION
MY_APP_NAME   MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME
MY_APP_NAME   8080
```

Pod のルート情報は、アクセスに使用するベース URL を提供します。上記の例では、アプリケーションにアクセス する ためにベース URL として http://MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME を使用します。

5.3.4. OpenShift Container Platform への Relational Database バックエンドサンプルアプリケーションのデプロイ

サンプルアプリケーションを OpenShift Container Platform に作成し、デプロイするプロセスは OpenShift Online と似ています。

前提条件

- developers.redhat.com/launch を使用して作成されたサンプルアプリケーション。

手順

- の手順に従って「OpenShift Online への Relational Database バックエンドサンプルアプリケーションのデプロイ」、OpenShift Container Platform Web コンソールからの URL およびユーザー認証情報のみを使用します。

5.3.5. Node.js のデータベースバックエンド API との対話

サンプルアプリケーションの作成が完了したら、以下のように対話できます。

前提条件

- アプリケーションを実行している。

- curl バイナリーまたは Web ブラウザー

手順

1. 以下のコマンドを実行して、アプリケーションの URL を取得します。

```
$ oc get route MY_APP_NAME
```

```
NAME            HOST/PORT                                   PATH      SERVICES        PORT
TERMINATION
MY_APP_NAME     MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME
MY_APP_NAME     8080
```

2. データベースアプリケーションの Web インターフェースにアクセスするには、ブラウザーで アプリケーション URL に移動します。

```
http://MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME
```

また、curl を使用して api/fruits/* エンドポイントで要求を直接実行できます。

データベースのすべてのエントリーを一覧表示します。

```
$ curl http://MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME/api/fruits
```

```
[ {
  "id" : 1,
  "name" : "Apple",
  "stock" : 10
}, {
  "id" : 2,
  "name" : "Orange",
  "stock" : 10
}, {
  "id" : 3,
  "name" : "Pear",
  "stock" : 10
} ]
```

特定の ID を持つエントリーの取得

```
$ curl http://MY_APP_NAME-
MY_PROJECT_NAME.OPENSHIFT_HOSTNAME/api/fruits/3
```

```
{
  "id" : 3,
  "name" : "Pear",
  "stock" : 10
}
```

新しいエントリーを作成します。

```
$ curl -H "Content-Type: application/json" -X POST -d '{"name":"Peach","stock":1}'
http://MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME/api/fruits
```

```
{
  "id" : 4,
  "name" : "Peach",
  "stock" : 1
}
```

エントリーを更新します。

```
$ curl -H "Content-Type: application/json" -X PUT -d '{"name":"Apple","stock":100}'
http://MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME/api/fruits/1
```

```
{
  "id" : 1,
  "name" : "Apple",
  "stock" : 100
}
```

エントリーを削除します。

```
$ curl -X DELETE http://MY_APP_NAME-
MY_PROJECT_NAME.OPENSHIFT_HOSTNAME/api/fruits/1
```

トラブルシューティング

- コマンドの実行後に HTTP エラーコード 503 を応答として受信すると、アプリケーションはまだ準備状態にならないことを意味します。

5.3.6. リレーショナルデータベースリソース

OpenShift でのリレーショナルデータベースの実行に関する背景および関連情報、CRUD、HTTP API、および REST に関する詳細は、こちらを参照してください。

- HTTP 動詞
- アーキテクチャーおよびネットワークベースのソフトウェアアーキテクチャーの設計 - Representational State Transfer(REST)
- 終了しない REST API 設計のデベース
- REST API は Hypertext 駆動型である必要があります
- parkardson Maturity Model
- 表現 Web Framework
- Spring Boot のリレーショナルデータベースバックエンド
- Eclipse Vert.x のリレーショナルデータベースバックエンド

- Thorntail のリレーショナルデータベースバックエンド

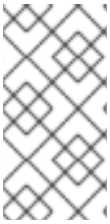
5.4. NODE.JS のヘルスチェック例

重要

以下の例は、実稼働環境で実行することは意図されていません。

実験レベルの例： Foundational。

アプリケーションをデプロイする場合、利用可能かどうか、および受信リクエストの処理を開始できるかどうかを確認することが重要になります。ヘルスチェックパターンを実装 すると、アプリケーションの正常 性を監視できます。これには、アプリケーションが利用可能かどうかや、要求に対応するかどうかが含まれます。



注記

ヘルスチェックの用語に精通していない場合には、まず「ヘルスチェックの概念」セクションを参照してください。

このユースケースの目的は、プローブを使用してヘルスチェックパターンを実証することです。プローブは、アプリケーションの liveness および readiness を報告するために使用されます。このユースケースでは、HTTP ヘルスエンドポイント を公開するアプリケーションを設定し、HTTP リクエストを発行します。コンテナーが正常であれば、ヘルス HTTP エンドポイントの liveness プローブにより、管理プラットフォームは 200 をリターンコードとして受け取り、これ以外のアクションは必要ありません。正常 HTTP エンドポイントが（スレッドがブロックされた場合など）応答を返しない場合、アプリケーションは liveness プローブに従って有効であるとみなされません。この場合、プラットフォームはそのアプリケーションに対応する Pod を強制終了し、新しい Pod を再作成し、アプリケーションを再起動します。

このユースケースでは、readiness プローブを実証し、使用することもできます。アプリケーションが稼働中で、再起動時にアプリケーションが HTTP 503 応答コードを返す場合などにリクエストを処理できない場合、このアプリケーションは readiness プローブに従って準備状態にあると見なされません。アプリケーションが readiness プローブによって準備状態とみなされない場合、要求はそのアプリケーションにルーティングされず、readiness プローブにより ready と見なされます。

5.4.1. ヘルスチェックの概念

ヘルスチェックパターンを理解するには、まず以下の概念を理解する必要があります。

Liveness

liveness は、アプリケーションが実行中かどうかを定義します。実行中のアプリケーションが応答しない状態または停止状態に移行する場合があり、再起動が必要になる場合があります。liveness の確認は、アプリケーションを再起動する必要があります。

準備状態 (Readines)

readiness は、実行中のアプリケーションが要求を提供できるかどうかを定義します。実行中のアプリケーションがエラーや破損状態に遷移する場合があり、要求に対応できなくなった可能性があります。readiness チェックは、要求がアプリケーションにルーティングされるかどうかを判断するのに役立ちます。

fail-over

フェイルオーバーにより、要求の処理の失敗を適切に処理できます。アプリケーションがリクエストのサービスを提供できない場合、その要求と今後のリクエストは 失敗したり、別の アプリケーションにルーティングしたりすることができます。これは通常、同じアプリケーションの冗長コピーです。

回復性と安定性

回復性と安定性により、要求への対応の失敗が適切に処理できるようになります。接続損失によるアプリケーションによるリクエストのサービス拒否に失敗した場合、接続の再確立後に要求を再試行できる回復性のあるシステムで、接続が失われてしまいます。

probe

プローブは実行中のコンテナーで定期的に実行する Kubernetes の動作です。

5.4.2. Health Check サンプルアプリケーションの OpenShift Online へのデプロイ

以下のオプションのいずれかを使用して、OpenShift Online で Health Check サンプルアプリケーションを実行します。

- developers.redhat.com/launch の使用
- oc CLI クライアントの使用

各方法は同じ oc コマンドを使用してアプリケーションをデプロイしますが、developers.redhat.com/launch を使用すると、oc コマンドを実行する自動デプロイメントワークフ

ローが提供されます。

5.4.2.1. developers.redhat.com/launch を使用したサンプルアプリケーションのデプロイ

このセクションでは、REST API Level 0 のサンプルアプリケーションをビルドし、Red Hat Developer Launcher Web インターフェースから OpenShift にデプロイする方法を説明します。

前提条件

- OpenShift Online のアカウント。

手順

1. ブラウザーで developers.redhat.com/launch URL に移動します。
2. 画面の指示に従って、Node.js でサンプルアプリケーションを作成して起動します。

5.4.2.2. oc CLI クライアントの認証

oc コマンドラインクライアントを使用して OpenShift Online でアプリケーションのサンプルを使用するには、OpenShift Online Web インターフェースによって提供されるトークンを使用してクライアントを認証する必要があります。

前提条件

- OpenShift Online のアカウント。

手順

1. ブラウザーで OpenShift Online URL に移動します。
2. Web コンソールの右上隅にある疑問符アイコンをクリックします。
3. ドロップダウンメニューで Command Line Tools を選択します。

4. oc login コマンドをコピーします。

5. 端末にコマンドを貼り付けます。このコマンドは、認証トークンを使用して CLI クライアント oc を OpenShift Online アカウントで認証します。

```
$ oc login OPENSHIFT_URL --token=MYTOKEN
```

5.4.2.3. oc CLI クライアントを使用した Health Check サンプルアプリケーションのデプロイ

このセクションでは、コマンドラインから Health Check サンプルアプリケーションをビルドし、これを OpenShift にデプロイする方法を説明します。

前提条件

- developers.redhat.com/launch を使用して作成されたサンプルアプリケーション。詳細はを参照してください 「developers.redhat.com/launch を使用したサンプルアプリケーションのデプロイ」。
- 認証された oc クライアント。詳細はを参照してください 「oc CLI クライアントの認証」。

手順

1. GitHub からプロジェクトのクローンを作成します。

```
$ git clone git@github.com:USERNAME/MY_PROJECT_NAME.git
```

プロジェクトの ZIP ファイルをダウンロードした場合は、展開します。

```
$ unzip MY_PROJECT_NAME.zip
```

2. 新しい OpenShift プロジェクトを作成します。

```
$ oc new-project MY_PROJECT_NAME
```

3. アプリケーションのルートディレクトリーに移動します。

4. npm を使用して OpenShift へのデプロイメントを開始します。

```
$ npm install && npm run openshift
```

これらのコマンドは、不足しているモジュール依存関係をすべてインストールし、Nodeshift モジュールを使用してサンプルアプリケーションを OpenShift にデプロイします。

5. アプリケーションのステータスを確認し、Pod が実行されていることを確認します。

```
$ oc get pods -w
NAME                      READY    STATUS     RESTARTS   AGE
MY_APP_NAME-1-aaaaa            1/1      Running    0          58s
MY_APP_NAME-s2i-1-build        0/1      Completed  0          2m
```

MY_APP_NAME-1-aaaaa Pod の完全なデプロイおよび起動後に、ステータスが Running である必要があります。また、続行する前に Pod の準備ができるまで待機する必要があります。これは READY 列に表示されます。たとえば、READY 列が 1 / 1 の場合、MY_APP_NAME-1-aaaaa の準備ができています。特定の Pod 名が異なります。新規ビルドごとに、中程度の数字が増加します。末尾の文字は、Pod の作成時に生成されます。

6. サンプルアプリケーションがデプロイされ、起動したら、そのルートを決定します。

ルート情報の例

```
$ oc get routes
NAME              HOST/PORT                                        PATH      SERVICES
PORT      TERMINATION
MY_APP_NAME       MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME
MY_APP_NAME      8080
```

Pod のルート情報は、アクセスに使用するベース URL を提供します。上記の例では、アプリケーションにアクセス する ためにベース URL として http://MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME を使用します。

5.4.3. Health Check サンプルアプリケーションの Minishift または CDK へのデプロイ

以下のいずれかのオプションを使用して、Minishift または CDK で Health Check サンプルアプリケーションをローカルに実行します。

- Fabric8 Launcher の使用
- oc CLI クライアントの使用

各方法は同じ oc コマンドを使用してアプリケーションをデプロイしますが、Fabric8 Launcher を使用すると oc コマンドを実行する自動デプロイメントワークフローが提供されます。

5.4.3.1. Fabric8 Launcher ツールの URL および認証情報の取得

Minishift または CDK でサンプルアプリケーションを作成してデプロイするには、Fabric8 Launcher ツール URL およびユーザー認証情報が必要です。この情報は、Minishift または CDK の起動時に提供されます。

前提条件

- Fabric8 Launcher ツールがインストールされ、設定されている。

手順

1. Minishift または CDK を起動したコンソールに移動します。
2. 実行中の Fabric8 Launcher にアクセスするために使用できる URL およびユーザー認証情報のコンソール出力を確認します。

Minishift または CDK 起動からのコンソール出力の例

```
...
-- Removing temporary directory ... OK
-- Server Information ...
   OpenShift server started.
   The server is accessible via web console at:
      https://192.168.42.152:8443

   You are logged in as:
```

```
    User:     developer
    Password: developer

To login as administrator:
    oc login -u system:admin
```

5.4.3.2. Fabric8 Launcher ツールを使用したサンプルアプリケーションのデプロイ

本セクションでは、REST API レベル 0 サンプルアプリケーションをビルドし、それを Fabric8 Launcher Web インターフェースから OpenShift にデプロイする方法を説明します。

前提条件

- 実行中の Fabric8 Launcher インスタンスの URL と Minishift または CDK のユーザー認証情報。詳細はを参照してください 「Fabric8 Launcher ツールの URL および認証情報の取得」。

手順

1. ブラウザーで Fabric8 Launcher URL に移動します。
2. 画面の指示に従って、Node.js でサンプルアプリケーションを作成して起動します。

5.4.3.3. oc CLI クライアントの認証

oc コマンドラインクライアントを使用して Minishift または CDK でアプリケーションの例を使用するには、Minishift または CDK Web インターフェースが提供するトークンを使用してクライアントを認証する必要があります。

前提条件

- 実行中の Fabric8 Launcher インスタンスの URL と Minishift または CDK のユーザー認証情報。詳細はを参照してください 「Fabric8 Launcher ツールの URL および認証情報の取得」。

手順

1.

1. ブラウザーで Minishift または CDK URL に移動します。

2. Web コンソールの右上隅にある疑問符アイコンをクリックします。

3. ドロップダウンメニューで Command Line Tools を選択します。

4. oc login コマンドをコピーします。

5. 端末にコマンドを貼り付けます。このコマンドは、認証トークンを使用して Minishift または CDK アカウントで oc CLI クライアントを認証します。

```
$ oc login OPENSHIFT_URL --token=MYTOKEN
```

5.4.3.4. oc CLI クライアントを使用した Health Check サンプルアプリケーションのデプロイ

このセクションでは、コマンドラインから Health Check サンプルアプリケーションをビルドし、これを OpenShift にデプロイする方法を説明します。

前提条件

- Minishift または CDK で Fabric8 Launcher ツールを使用して作成されたアプリケーションのサンプル。詳細はを参照してください 「Fabric8 Launcher ツールを使用したサンプルアプリケーションのデプロイ」。
- Fabric8 Launcher ツール URL。
- 認証された oc クライアント。詳細はを参照してください 「oc CLI クライアントの認証」。

手順

1. GitHub からプロジェクトのクローンを作成します。

```
$ git clone git@github.com:USERNAME/MY_PROJECT_NAME.git
```

プロジェクトの ZIP ファイルをダウンロードした場合は、展開します。

```
$ unzip MY_PROJECT_NAME.zip
```

2. 新しい OpenShift プロジェクトを作成します。

```
$ oc new-project MY_PROJECT_NAME
```

3. アプリケーションのルートディレクトリーに移動します。

4. npm を使用して OpenShift へのデプロイメントを開始します。

```
$ npm install && npm run openshift
```

これらのコマンドは、不足しているモジュール依存関係をすべてインストールし、Nodeshift モジュールを使用してサンプルアプリケーションを OpenShift にデプロイします。

5. アプリケーションのステータスを確認し、Pod が実行されていることを確認します。

```
$ oc get pods -w
NAME                     READY    STATUS     RESTARTS   AGE
MY_APP_NAME-1-aaaaa            1/1      Running    0          58s
MY_APP_NAME-s2i-1-build        0/1      Completed  0          2m
```

MY_APP_NAME-1-aaaaa Pod の完全なデプロイおよび起動後に、ステータスが Running である必要があります。また、続行する前に Pod の準備ができるまで待機する必要があります。これは READY 列に表示されます。たとえば、READY 列が 1 / 1 の場合、MY_APP_NAME-1-aaaaa の準備ができています。特定の Pod 名が異なります。新規ビルドごとに、中程度の数字が増加します。末尾の文字は、Pod の作成時に生成されます。

6. サンプルアプリケーションがデプロイされ、起動したら、そのルートを決定します。

ルート情報の例

```
$ oc get routes
```

```
NAME              HOST/PORT                                    PATH      SERVICES
PORT      TERMINATION
MY_APP_NAME       MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME
MY_APP_NAME      8080
```

Pod のルート情報は、アクセスに使用するベース URL を提供します。上記の例では、アプリケーションにアクセス する ためにベース URL として http://MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME を使用します。

5.4.4. Health Check サンプルアプリケーションの OpenShift Container Platform へのデプロイ

サンプルアプリケーションを OpenShift Container Platform に作成し、デプロイするプロセスは OpenShift Online と似ています。

前提条件

- developers.redhat.com/launch を使用して作成されたサンプルアプリケーション。

手順

- の手順に従って「Health Check サンプルアプリケーションの OpenShift Online へのデプロイ」、OpenShift Container Platform Web コンソールからの URL およびユーザー認証情報のみを使用します。

5.4.5. 変更されていないヘルスチェックサンプルアプリケーションとの対話

サンプルのアプリケーションをデプロイすると、MY_APP_NAME サービスが実行されます。MY_APP_NAME サービスは以下の REST エンドポイントを公開します。

/api/greeting

name パラメーターの greeting が含まれる JSON を返します（またはデフォルト値として World）。

/api/stop

障害をシミュレートするために、サービスが応答しなくなるように強制します。

以下の手順は、サービスの可用性を検証し、障害をシミュレートする方法を説明します。利用可能なサービスが失敗すると、OpenShift の自己修復機能がサービスでトリガーされます。

Web インターフェースを使用して、この手順を実行できます。

1. **curl を使用して MY_APP_NAME サービスに対して GET 要求を実行します。ブラウザーを使用してこれを行うこともできます。**

```
$ curl http://MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME/api/greeting
```

```
{"content":"Hello, World!"}
```

2. **/api/stop エンドポイントを呼び出して、その直後に /api/greeting エンドポイントの可用性を確認します。**

/api/stop エンドポイントを呼び出すと、内部サービスの失敗をシミュレートし、OpenShift の自己修復機能がトリガーされます。障害発生後に /api/greeting を呼び出すと、サービスは HTTP ステータス 503 を返すはずです。

```
$ curl http://MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME/api/stop
```

```
Stopping HTTP server, Bye bye world !
```

（以下で説明）

```
$ curl http://MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME/api/greeting
```

```
Not online
```

3. **oc get pods -w を使用して、自己修復機能の動作を継続的に監視します。**

サービスの障害を呼び出す際には、OpenShift コンソールでの自己修復機能、または oc クライアントツールを使って自己修復機能を確認できます。READY 状態の Pod 数はゼロ(0 /1) に移行し、短い期間（1 分未満）が最大1 /1に移行します。さらに、サービスが失敗するたびに RESTARTS の 数 が増加します。

```
$ oc get pods -w
NAME                    READY    STATUS    RESTARTS   AGE
```

```
MY_APP_NAME-1-26iy7   0/1      Running   5         18m
MY_APP_NAME-1-26iy7   1/1      Running   5         19m
```

4. **オプション： Web インターフェースを使用してサービスを呼び出します。**

ターミナルウィンドウを使用した対話の結果として、サービスが提供する Web インターフェースを使用して、異なるメソッドを呼び出すと、ライフサイクルフェーズでサービスの動作を確認できます。

```
http://MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME
```

5. **オプション： Web コンソールを使用して、セルフサービスプロセスの各段階でアプリケーションが生成したログ出力を表示します。**

   1. **プロジェクトに移動します。**
   2. **サイドバーで Monitoring をクリックします。**
   3. **画面の右上隅にある Events をクリックしてログメッセージを表示します。**
   4. **オプション： View Details をクリックし、イベントログの詳細ビューを表示します。**

ヘルスチェックアプリケーションは以下のメッセージを生成します。

メッセージ	Status
Unhealthy	readiness プローブに失敗しました。このメッセージは予想され、**/api/greeting** エンドポイントのシミュレーションの失敗が検出され、自己修復プロセスが開始されます。
Killing	サービスを実行している利用できない Docker コンテナーは、再作成する前に強制終了されます。
Pulling	最新バージョンの Docker イメージをダウンロードして、コンテナーを再作成します。

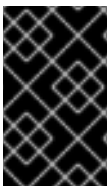
メッセージ	Status
Pulled	Docker イメージが正常にダウンロードされました。
Created	Docker コンテナーが正常に作成されました。
Started	Docker コンテナーが要求を処理する準備ができています。

5.4.6. ヘルスチェックのリソース

ヘルスチェックに関する背景や関連情報は、以下を参照してください。

- OpenShift の Application Health
- Kubernetes liveness and Readiness プローブ
- Spring Boot のヘルスチェック
- Health Check for Eclipse Vert.x（Eclipse Vert.x のヘルスチェック）
- Health Check for Thorntail

5.5. NODE.JS のサーキットブレーカーの例



重要

以下の例は、実稼働環境で実行することは意図されていません。

制限： Minishift または CDK でこのサンプルアプリケーションを実行します。手動でのワークフローを使用して、この例を OpenShift Online Pro および OpenShift Container Platform にデプロイすることもできます。この例では、現在 OpenShift Online ドキュメンテーションでは使用できません。

実験レベルの例： Foundational。

Circuit Breaker の例は、サービスの失敗を報告し、要求の処理に利用できるまで失敗したサービスへのアクセスを制限します。これにより、機能障害の発生したサービスに依存する他のサービスでエラーが発生するのを防ぐことができます。

以下の例では、サービスに Circuit Breaker および Fallback パターンを実装する方法を説明します。

5.5.1. サーキットブレーカー設計パターン

Circuit Breaker は以下の目的とするパターンです。

- サービスが他のサービスを同期的に呼び出すサービスアーキテクチャーにおける、ネットワーク障害の影響やレイテンシーが高くなる。

  いずれかのサービスがある場合：

  - ネットワーク障害により利用不可になる
  - トラフィックの難読化により、レイテンシーの値がかなり長くなります。

  エンドポイントの呼び出しを試みる他のサービスは、到達を試みる際に重要なリソースを使い果たしなくなり、レンダリングが不可能になる可能性があります。

- マイクロサービスアーキテクチャー全体でレンダリングできないように、コンピューター障害としても知られるようにします。

- 保護された機能とリモート機能の間のプロキシーとして動作し、障害を監視するリモート機能として機能します。

- エラーが特定のしきい値に到達し、サーキットブレーカーへの呼び出しがすべて、保護された呼び出しを行わずに、エラーや事前定義のフォールバック応答を返します。

サーキットブレーカーには、通常、サーキットブレーカーをトリップしたときに通知を行うエラー報告メカニズムも含まれています。

サーキットブレーカーの実装

- Circuit Breaker パターンを実装すると、サービスクライアントは一定の間隔でプロキシー経由でリモートサービスエンドポイントを呼び出します。
- リモートサービスエンドポイントへの呼び出しが繰り返し失敗し、一貫して Circuit Breaker トリップを行うと、サービスへのすべての呼び出しは設定されたタイムアウト期間を即座に失敗し、事前定義されたフォールバック応答を返します。
- タイムアウトの期限が切れると、リモートサービスに渡すテストコールの数に制限され、16 進法か、使用できなくなったかを判断できます。
  - テスト呼び出しが失敗すると、サーキットブレーカーはサービスを利用できず、受信呼び出しにフォールバック応答を返します。
  - テスト呼び出しに成功すると、サーキットブレーカーが終了し、トラフィックが再びリモートサービスに到達できるようになります。

5.5.2. サーキットブレーカー設計のトレードオフ

表5.4 Design Tradeoffs

pros	cons
● サービスが呼び出す他のサービスの障害を処理できるようにします。	● タイムアウト値の最適化が困難な場合があります。 ○ 超過分のタイムアウト値が長いと、過剰なレイテンシーが発生することがあります。 ○ 小さい方のタイムアウト値が小さいと、誤検出が発生する可能性があります。

5.5.3. サーキットブレーカーアプリケーションの OpenShift Online へのデプロイ

以下のオプションのいずれかを使用して、OpenShift Online で Circuit Breaker サンプルアプリケーションを実行します。

- developers.redhat.com/launch の使用
- oc CLI クライアントの使用

各方法は同じ oc コマンドを使用してアプリケーションをデプロイしますが、developers.redhat.com/launch を使用すると、oc コマンドを実行する自動デプロイメントワークフローが提供されます。

5.5.3.1. developers.redhat.com/launch を使用したサンプルアプリケーションのデプロイ

このセクションでは、REST API Level 0 のサンプルアプリケーションをビルドし、Red Hat Developer Launcher Web インターフェースから OpenShift にデプロイする方法を説明します。

前提条件

- OpenShift Online のアカウント。

手順

1. ブラウザーで developers.redhat.com/launch URL に移動します。
2. 画面の指示に従って、Node.js でサンプルアプリケーションを作成して起動します。

5.5.3.2. oc CLI クライアントの認証

oc コマンドラインクライアントを使用して OpenShift Online でアプリケーションのサンプルを使用するには、OpenShift Online Web インターフェースによって提供されるトークンを使用してクライアントを認証する必要があります。

前提条件

- OpenShift Online のアカウント。

手順

1. ブラウザーで OpenShift Online URL に移動します。

2. Web コンソールの右上隅にある疑問符アイコンをクリックします。

3. ドロップダウンメニューで Command Line Tools を選択します。

4. oc login コマンドをコピーします。

5. 端末にコマンドを貼り付けます。このコマンドは、認証トークンを使用して CLI クライアント oc を OpenShift Online アカウントで認証します。

```
$ oc login OPENSHIFT_URL --token=MYTOKEN
```

5.5.3.3. oc CLI クライアントを使用した Circuit Breaker サンプルアプリケーションのデプロイ

本セクションでは、Circuit Breaker サンプルアプリケーションをビルドし、コマンドラインから OpenShift にデプロイする方法を説明します。

前提条件

- developers.redhat.com/launch を使用して作成されたサンプルアプリケーション。詳細はを参照してください 「developers.redhat.com/launch を使用したサンプルアプリケーションのデプロイ」。
- 認証された oc クライアント。詳細はを参照してください 「oc CLI クライアントの認証」。

手順

1. GitHub からプロジェクトのクローンを作成します。

```
$ git clone git@github.com:USERNAME/MY_PROJECT_NAME.git
```

プロジェクトの ZIP ファイルをダウンロードした場合は、展開します。

```
$ unzip MY_PROJECT_NAME.zip
```

2. 新しい OpenShift プロジェクトを作成します。

```
$ oc new-project MY_PROJECT_NAME
```

3. アプリケーションのルートディレクトリーに移動します。

4. 提供される start-openshift.sh スクリプトを使用して、OpenShift へのデプロイメントを開始します。

```
$ chmod +x start-openshift.sh
$ ./start-openshift.sh
```

これらのコマンドは Nodeshift npm モジュールを使用して依存関係をインストールし、S2I ビルドプロセスを OpenShift で起動し、サービスを起動します。

5. アプリケーションのステータスを確認し、Pod が実行されていることを確認します。

```
$ oc get pods -w
NAME                          READY     STATUS      RESTARTS   AGE
MY_APP_NAME-greeting-1-aaaaa     1/1       Running   0          17s
MY_APP_NAME-greeting-1-deploy    0/1       Completed 0          22s
MY_APP_NAME-name-1-aaaaa         1/1       Running   0          14s
MY_APP_NAME-name-1-deploy        0/1       Completed 0          28s
```

MY_APP_NAME-greeting-1-aaaaa および MY_APP_NAME-name-1-aaaaa Pod のステータスは、完全にデプロイされ、起動すると Running でなければなりません。また、続行する前に Pod の準備ができるまで待機する必要があります。これは READY 列に表示されます。たとえば、READY 列が 1 / 1 の場合、MY_APP_NAME-greeting-1-aaaaa が準備状態となります。特定の Pod 名が異なります。新規ビルドごとに、中程度の数字が増加します。末尾の文字は、Pod の作成時に生成されます。

6. サンプルアプリケーションがデプロイされ、起動したら、そのルートを決定します。

ルート情報の例

```
$ oc get routes
```

```
NAME                HOST/PORT                                   PATH      SERVICES
PORT      TERMINATION
MY_APP_NAME-greeting   MY_APP_NAME-greeting-
MY_PROJECT_NAME.OPENSHIFT_HOSTNAME          MY_APP_NAME-greeting   8080
None
MY_APP_NAME-name       MY_APP_NAME-name-
MY_PROJECT_NAME.OPENSHIFT_HOSTNAME          MY_APP_NAME-name       8080
None
```

Pod のルート情報は、アクセスに使用するベース URL を提供します。上記の例では、アプリケーションにアクセス する ためにベース URL として http://MY_APP_NAME-greeting-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME を使用します。

5.5.4. サーキットブレーカーサンプルアプリケーションの Minishift または CDK へのデプロイ

以下のいずれかのオプションを使用して、Minishift または CDK で Circuit Breaker サンプルアプリケーションをローカルに実行します。

- Fabric8 Launcher の使用
- oc CLI クライアントの使用

各方法は同じ oc コマンドを使用してアプリケーションをデプロイしますが、Fabric8 Launcher を使用すると oc コマンドを実行する自動デプロイメントワークフローが提供されます。

5.5.4.1. Fabric8 Launcher ツールの URL および認証情報の取得

Minishift または CDK でサンプルアプリケーションを作成してデプロイするには、Fabric8 Launcher ツール URL およびユーザー認証情報が必要です。この情報は、Minishift または CDK の起動時に提供されます。

前提条件

- Fabric8 Launcher ツールがインストールされ、設定されている。

手順

1. Minishift または CDK を起動したコンソールに移動します。

2. 実行中の Fabric8 Launcher にアクセスするために使用できる URL およびユーザー認証情報のコンソール出力を確認します。

Minishift または CDK 起動からのコンソール出力の例

```
...
-- Removing temporary directory ... OK
-- Server Information ...
   OpenShift server started.
   The server is accessible via web console at:
      https://192.168.42.152:8443

   You are logged in as:
      User:     developer
      Password: developer

   To login as administrator:
      oc login -u system:admin
```

5.5.4.2. Fabric8 Launcher ツールを使用したサンプルアプリケーションのデプロイ

本セクションでは、REST API レベル 0 サンプルアプリケーションをビルドし、それを Fabric8 Launcher Web インターフェースから OpenShift にデプロイする方法を説明します。

前提条件

- 実行中の Fabric8 Launcher インスタンスの URL と Minishift または CDK のユーザー認証情報。詳細はを参照してください 「Fabric8 Launcher ツールの URL および認証情報の取得」。

手順

1. ブラウザーで Fabric8 Launcher URL に移動します。

2. 画面の指示に従って、Node.js でサンプルアプリケーションを作成して起動します。

5.5.4.3. oc CLI クライアントの認証

oc コマンドラインクライアントを使用して Minishift または CDK でアプリケーションの例を使用するには、Minishift または CDK Web インターフェースが提供するトークンを使用してクライアントを認証する必要があります。

前提条件

- 実行中の Fabric8 Launcher インスタンスの URL と Minishift または CDK のユーザー認証情報。詳細はを参照してください 「Fabric8 Launcher ツールの URL および認証情報の取得」。

手順

1. ブラウザーで Minishift または CDK URL に移動します。
2. Web コンソールの右上隅にある疑問符アイコンをクリックします。
3. ドロップダウンメニューで Command Line Tools を選択します。
4. oc login コマンドをコピーします。
5. 端末にコマンドを貼り付けます。このコマンドは、認証トークンを使用して Minishift または CDK アカウントで oc CLI クライアントを認証します。

```
$ oc login OPENSHIFT_URL --token=MYTOKEN
```

5.5.4.4. oc CLI クライアントを使用した Circuit Breaker サンプルアプリケーションのデプロイ

本セクションでは、Circuit Breaker サンプルアプリケーションをビルドし、コマンドラインから OpenShift にデプロイする方法を説明します。

前提条件

- Minishift または CDK で Fabric8 Launcher ツールを使用して作成されたアプリケーションのサンプル。詳細はを参照してください 「Fabric8 Launcher ツールを使用したサンプルアプリケーションのデプロイ」。

- Fabric8 Launcher ツール URL。
- 認証された oc クライアント。詳細はを参照してください 「oc CLI クライアントの認証」。

手順

1. GitHub からプロジェクトのクローンを作成します。

```
$ git clone git@github.com:USERNAME/MY_PROJECT_NAME.git
```

プロジェクトの ZIP ファイルをダウンロードした場合は、展開します。

```
$ unzip MY_PROJECT_NAME.zip
```

2. 新しい OpenShift プロジェクトを作成します。

```
$ oc new-project MY_PROJECT_NAME
```

3. アプリケーションのルートディレクトリーに移動します。

4. 提供される start-openshift.sh スクリプトを使用して、OpenShift へのデプロイメントを開始します。

```
$ chmod +x start-openshift.sh
$ ./start-openshift.sh
```

これらのコマンドは Nodeshift npm モジュールを使用して依存関係をインストールし、S2I ビルドプロセスを OpenShift で起動し、サービスを起動します。

5. アプリケーションのステータスを確認し、Pod が実行されていることを確認します。

```
$ oc get pods -w
NAME                          READY     STATUS      RESTARTS   AGE
MY_APP_NAME-greeting-1-aaaaa     1/1       Running   0          17s
```

```
MY_APP_NAME-greeting-1-deploy    0/1       Completed 0          22s
MY_APP_NAME-name-1-aaaaa         1/1       Running   0          14s
MY_APP_NAME-name-1-deploy        0/1       Completed 0          28s
```

MY_APP_NAME-greeting-1-aaaaa および MY_APP_NAME-name-1-aaaaa Pod のステータスは、完全にデプロイされ、起動すると Running でなければなりません。また、続行する前に Pod の準備ができるまで待機する必要があります。これは READY 列に表示されます。たとえば、READY 列が 1 / 1 の場合、MY_APP_NAME-greeting-1-aaaaa が準備状態となります。特定の Pod 名が異なります。新規ビルドごとに、中程度の数字が増加します。末尾の文字は、Pod の作成時に生成されます。

6. サンプルアプリケーションがデプロイされ、起動したら、そのルートを決定します。

ルート情報の例

```
$ oc get routes
NAME                HOST/PORT                                        PATH      SERVICES
PORT      TERMINATION
MY_APP_NAME-greeting   MY_APP_NAME-greeting-
MY_PROJECT_NAME.OPENSHIFT_HOSTNAME          MY_APP_NAME-greeting   8080
None
MY_APP_NAME-name       MY_APP_NAME-name-
MY_PROJECT_NAME.OPENSHIFT_HOSTNAME          MY_APP_NAME-name       8080
None
```

Pod のルート情報は、アクセスに使用するベース URL を提供します。上記の例では、アプリケーションにアクセス する ためにベース URL として http://MY_APP_NAME-greeting-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME を使用します。

5.5.5. サーキットブレーカーサンプルアプリケーションの OpenShift Container Platform へのデプロイ

サンプルアプリケーションを OpenShift Container Platform に作成し、デプロイするプロセスは OpenShift Online と似ています。

前提条件

- developers.redhat.com/launch を使用して作成されたサンプルアプリケーション。

手順

- の手順に従って「サーキットブレーカーアプリケーションの OpenShift Online へのデプロイ」、OpenShift Container Platform Web コンソールからの URL およびユーザー認証情報のみを使用します。

5.5.6. 未変更の Node.js Circuit Breaker サンプルアプリケーションとの対話

Node.js サンプルアプリケーションをデプロイしたら、以下のサービスが実行されます。

MY_APP_NAME-name

以下のエンドポイントを公開します。

- このサービスの機能時に 名前を返す /api/ name エンドポイントと、このサービスが失敗を示すよう設定されているとエラーが返されます。
- /api/ name エンドポイントの 動作を制御し、サービスが正しく動作するか、またはエラーを実証する /api/ state エンドポイント。

MY_APP_NAME-greeting

以下のエンドポイントを公開します。

- /api/greeting エンドポイント。 このエンドポイントは、あいまいな応答を取得するために呼び出しできます。

  /api/greeting エンドポイントを呼び出すと、リクエストの処理の一環として MY_APP_NAME- name サービスの /api/ name エンドポイントに対する呼び出しが発行されます。/api/name エンドポイントに対して行われる呼び出しは、サーキットブレーカーによって保護されます。

  リモートエンドポイントが利用可能になると、name サービスは HTTP コード 200 (OK)で応答し、/api/greeting エンドポイントから以下の greeting を受け取ります。

  ```
  {"content":"Hello, World!"}
  ```

  リモートエンドポイントが利用できない場合、name サービスは HTTP コード 500（内部サーバーエラー）で応答し、/api/greeting エンドポイントから事前定義されたフォー

ルバック応答を受け取ります。

```
{"content":"Hello, Fallback!"}
```

- Circuit Breaker の 状態を返す /api/cb- state エンドポイント。状態は以下のとおりです。
  - Open : サーキットブレーカーは、失敗したサービスに到達できないようにします。
  - closed: サーキットブレーカーはサービスに到達できる要求を許可します。

以下の手順は、サービスの可用性を検証し、障害をシミュレートし、フォールバック応答を受け取る方法を示しています。

1. curl を使用して MY_APP_NAME-greeting サービスに対して GET 要求を実行します。Web インターフェースの Invoke ボタンを使用してこれを行うこともできます。

```
$ curl http://MY_APP_NAME-greeting-
MY_PROJECT_NAME.LOCAL_OPENSHIFT_HOSTNAME/api/greeting
{"content":"Hello, World!"}
```

2. MY_APP_NAME-name サービスの失敗をシミュレートするには、以下を実行します。
   - Web インターフェース の Toggle ボタンを使用します。
   - MY_APP_NAME-name サービスを実行している Pod のレプリカ数を 0 にスケールダウンします。
   - MY_APP_NAME-name サービスの /api/state エンドポイントに対して HTTP PUT 要求を実行し、その state を fail に設定し ます。

```
$ curl -X PUT -H "Content-Type: application/json" -d '{"state": "fail"}'
http://MY_APP_NAME-name-
MY_PROJECT_NAME.LOCAL_OPENSHIFT_HOSTNAME/api/state
```

3. /api/greeting エンドポイントを呼び出します。/api/name エンドポイントでの複数の要求が失敗する場合：

   a. サーキットブレーカーが開きます。

   b. Web インターフェースの状態インジケーターが CLOSED から OPEN に変更されます。

   c. サーキットブレーカーは、/api/greeting エンドポイントを呼び出すとフォールバック応答を発行します。

```
$ curl http://MY_APP_NAME-greeting-
MY_PROJECT_NAME.LOCAL_OPENSHIFT_HOSTNAME/api/greeting
{"content":"Hello, Fallback!"}
```

4. MY_APP_NAME-name サービスの可用性に復元します。そのためには、以下を行います。

   - Web インターフェース の Toggle ボタンを使用します。
   - MY_APP_NAME-name サービスを実行している Pod のレプリカ数を 1 にスケールアップします。
   - MY_APP_NAME-name サービスの /api/state エンドポイントに対して HTTP PUT 要求を実行し、その状態を ok に設定します。

```
$ curl -X PUT -H "Content-Type: application/json" -d '{"state": "ok"}'
http://MY_APP_NAME-name-
MY_PROJECT_NAME.LOCAL_OPENSHIFT_HOSTNAME/api/state
```

5. /api/greeting エンドポイントを再度呼び出します。/api/name エンドポイントでの複数の要求に成功すると、以下を実行します。

   a. サーキットブレーカーを閉じます。

   b.

Web インターフェースの状態インジケーターが OPEN から CLOSED に変更されます。

c. Circuit Breaker は、/api/greeting エンドポイントを呼び出すと Hello World! greeting を 返します。

```
$ curl http://MY_APP_NAME-greeting-MY_PROJECT_NAME.LOCAL_OPENSHIFT_HOSTNAME/api/greeting
{"content":"Hello, World!"}
```

5.5.7. サーキットブレーカーリソース

Circuit Breaker パターンの設計原理に関する背景情報は、以下のリンクに従います。

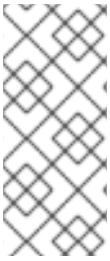
- microservices.io: Microservice Patterns: Circuit Breaker
- Martin Fowler: CircuitBreaker
- Spring Boot のサーキットブレーカー
- Eclipse Vert.x のサーキットブレーカー
- Thorntail のサーキットブレーカー

5.6. NODE.JS のセキュアなサンプルアプリケーション

重要

以下の例は、実稼働環境で実行することは意図されていません。

制限： Minishift または CDK でこのサンプルアプリケーションを実行します。手動でのワークフローを使用して、この例を OpenShift Online Pro および OpenShift Container Platform にデプロイすることもできます。この例では、現在 OpenShift Online ドキュメンテーションでは使用できません。



注記

Node.js の Secured サンプルアプリケーションには Red Hat SSO 7.3 が必要です。IBM Z では Red Hat SSO 7.3 に対応していないため、IBM Z では Secured の例は利用できません。

頻度レベルの例： 高度。

Secured サンプルアプリケーションは、Red Hat SSO を使用して REST エンドポイントを保護します。（この例では、REST API Level 0 の例を展開します。

Red Hat SSO:

- OAuth 2.0 仕様のエクステンションである Open ID Connect プロトコルを実装します。
- アクセストークンを発行し、セキュアなリソースにさまざまなアクセス権限をクライアントに提供します。

SSO を使用してアプリケーションのセキュリティーを保護すると、セキュリティー設定を一元化しながらアプリケーションのセキュリティーを追加できます。

重要

この例では、デモ目的で Red Hat SSO が事前設定されており、その原則、使用方法、または設定について説明しません。この例を使用する前に、Red Hat SSO に関する基本的な概念を理解していることを確認してください。

5.6.1. Secured プロジェクト構造

SSO の例には以下が含まれます。

- Greeting サービスのソース（セキュリティー保護の対象）

- SSO サーバーをデプロイするテンプレートファイル(service.sso.yaml)
- サービスをセキュアにする Keycloak アダプターの設定

5.6.2. Red Hat SSO デプロイメントの設定

この例の service.sso.yaml ファイルには、事前に設定された Red Hat SSO サーバーをデプロイするすべての OpenShift 設定項目が含まれています。この演習により SSO サーバー設定が簡素化され、事前設定されたユーザーとセキュリティー設定により追加設定なしで設定を行うことができます。service.sso.yaml ファイルには非常に長い行が含まれ、gedit などの一部のテキストエディターでは、このファイルの読み取りに問題がある場合があります。

警告

この SSO 設定を実稼働環境で使用することは推奨されません。特に、セキュリティー設定例に加えられた概念実証は、実稼働環境でそれを使用する機能に影響します。

表5.5 SSO 例の簡素化

変更	理由	推奨事項
デフォルト設定には、**yaml 設定ファイルの公開鍵と秘密鍵の両方が含まれます。**	これは、エンドユーザーは Red Hat SSO モジュールをデプロイでき、内部や Red Hat SSO の設定方法を確認せずに使用可能な状態になっているためです。	実稼働環境では、秘密鍵をソースコントロールに保存しないでください。サーバー管理者が追加する必要があります。
設定された **クライアントはコールバック URL を受け入れ** ます。	各ランタイムにカスタム設定を作成しないようにするには、OAuth2 仕様で必要なコールバックの検証を回避します。	アプリケーション固有のコールバック URL には、有効なドメイン名を指定する必要があります。
クライアントには SSL/TLS は必要ありませんが、セキュアなアプリケーションは HTTPS 経由で公開されません。	これらの例は、各ランタイムに生成される証明書を生成しないために単純化されます。	実稼働環境では、セキュアなアプリケーションでは、プレーン HTTP ではなく HTTPS を使用する必要があります。

変更	理由	推奨事項
デフォルトの 1分より、トークンのタイムアウトが 10 分に増えました。	コマンドラインの例を使用する際に、ユーザーエクスペリエンスを向上	セキュリティーの観点からすると、攻撃者はアクセストークンが拡張されていることを推測する必要があります。攻撃者が現在のトークンを推測するのにより困難となるため、このウィンドウを維持することが推奨されます。

5.6.3. Red Hat SSO レルムモデル

この例のセキュア化に は、マスターレルムを使用します。コマンドラインクライアントのモデルとセキュアな REST エンドポイントのモデルを提供する、事前に設定されたアプリケーションクライアント定義が 2 つあります。

Red Hat SSO マスターレルム には、admin と alice のさまざまな認証および承認の結果を検証するために使用できる事前に設定されたユーザーを 2 つあります。

5.6.3.1. Red Hat SSO ユーザー

セキュリティー保護されたサンプルのレルムモデルには、以下の 2 つのユーザーが含まれます。

admin

admin ユーザーには admin のパスワードがあり、レルムの管理者です。このユーザーは Red Hat SSO 管理コンソールに完全アクセスできますが、セキュアなエンドポイントへのアクセスに必要なロールマッピングはありません。このユーザーを使用して、認証されていないユーザーの動作を確認できます。

ディレクター

alice ユーザーにはパスワードがあり 、 正規のアプリケーションユーザーです。このユーザーは、認証され、セキュアなエンドポイントへの認証に成功したアクセスを示します。ロールマッピングの例は、以下のデコードされた JWT ベアラートークンに提供されます。

```
{
  "jti": "0073cfaa-7ed6-4326-ac07-c108d34b4f82",
  "exp": 1510162193,
  "nbf": 0,
  "iat": 1510161593,
  "iss": "https://secure-sso-sso.LOCAL_OPENSHIFT_HOSTNAME/auth/realms/master", (1)
  "aud": "demoapp",
  "sub": "c0175ccb-0892-4b31-829f-dda873815fe8",
  "typ": "Bearer",
```

```
  "azp": "demoapp",
  "nonce": "90ff5d1a-ba44-45ae-a413-50b08bf4a242",
  "auth_time": 1510161591,
  "session_state": "98efb95a-b355-43d1-996b-0abcb1304352",
  "acr": "1",
  "client_session": "5962112c-2b19-461e-8aac-84ab512d2a01",
  "allowed-origins": [
    "*"
  ],
  "realm_access": {
    "roles": [ 2
      "example-admin"
    ]
  },
  "resource_access": { 3
    "secured-example-endpoint": {
      "roles": [
        "example-admin" 4
      ]
    },
    "account": {
      "roles": [
        "manage-account",
        "view-profile"
      ]
    }
  },
  "name": "Alice lnChains",
  "preferred_username": "alice", 5
  "given_name": "Alice",
  "family_name": "lnChains",
  "email": "alice@keycloak.org"
}
```


1

iss フィールドは、トークンを発行する Red Hat SSO レルムインスタンス URL に対応します。これは、トークンを検証するには、セキュアなエンドポイントデプロイメントで設定する必要があります。

2

roles オブジェクトは、グローバルレルムレベルでユーザーに付与されたロールを提供します。この場合、alice には example-admin ロールが付与されています。セキュリティーが保護されたエンドポイントは、承認されたロールのレルムレベルを探します。

3

resource_access オブジェクトには、リソース固有のロール付与が含まれます。このオブジェクトの下に、セキュアなエンドポイントごとにオブジェクトを見つけます。

4

resource_access.secured-example-endpoint.roles オブジェクトには、secured-example-endpoint リソースの alice に付与さ れるロールが含まれます。

5

preferred_username フィールドは、アクセストークンの生成に使用されたユーザー名を提供します。

5.6.3.2. アプリケーションクライアント

OAuth 2.0 仕様では、リソースの所有者の代わりにセキュアなリソースにアクセスするアプリケーションクライアントのロールを定義できます。マスターレルムには、以下 のアプリケーションクライアントが 定義されます。

demoapp

これは、アクセストークンの取得に使用されるクライアントシークレットを持つ 機密 タイプクライアントです。トークンには、alice が Thorntail、Eclipse Vert.x 、 Node.js、および Spring Boot ベースのアプリケーションデプロイメントにアクセスできるようにする alice ユーザーの付与が含まれます。

secured-example-endpoint

secured-example-endpoint はベアラーのみのタイプのクライアントで、関連するリソース（とくに Greeting サービス）へのアクセスに example-admin ロールを必要とします。

5.6.4. Node.js SSO アダプターの設定

SSO アダプターは、Web リソースでセキュリティーを実施する、または SSO サーバーへのクライアント 側 またはクライアントです。この場合、これは Greeting サービスです。

セキュリティー例の Node.js コードの実行

```
const express = require('express');
const Keycloak = require('keycloak-connect'); 1
const kc = new Keycloak({}); 2

const app = express();
```

```
app.use(kc.middleware()); (3)

app.use('/api/greeting', kc.protect('example-admin'), callback); (4)
```

(1) npm モジュール keycloak-connect がインストールされ、必要 です。keycloak-connect モジュールは、表現 との統合を提供する 接続ミドルウェア として機能します。

(2) 新しい Keycloak オブジェクトをインスタンス化し、空の設定オブジェクトを渡します。

(3) Keycloak をミドルウェアとして使用するように 表現 します。

(4) リソースにアクセスする前に、ユーザーを認証し、example-admin ロールの一部に強制します。

keycloak.jsonを使用した Keycloak Adapter でのセキュリティーの実施

```
{
  "realm": "master", (1)
  "resource": "secured-example-endpoint", (2)
  "realm-public-key": "...", (3)
  "auth-server-url": "${env.SSO_AUTH_SERVER_URL}", (4)
  "ssl-required": "external",
  "disable-trust-manager": true,
  "bearer-only": true, (5)
  "use-resource-role-mappings": true
}
```

(1) 使用するセキュリティーレルム。

2

実際の Keycloak クライアント の設定。

3

レルム公開鍵の PEM 形式。これは管理コンソールから取得できます。

4

Red Hat SSO サーバーのアドレス（ビルド時の使用）。

5

有効にすると、ユーザーの認証は試行されず、ベアラートークンのみを確認します。

Node.js コードのサンプルは、Keycloak を有効にし、Greeting サービスの Web リソースエンドポイントの保護を実施します。keycloak.json は、セキュリティーアダプターが Red Hat SSO と対話するように設定します。

関連情報

- Node.js Keycloak アダプターの詳細は、Keycloak ドキュメントを参照して ください。

5.6.5. Minishift または CDK へのセキュアなサンプルアプリケーションのデプロイ

5.6.5.1. Fabric8 Launcher ツールの URL および認証情報の取得

Minishift または CDK でサンプルアプリケーションを作成してデプロイするには、Fabric8 Launcher ツール URL およびユーザー認証情報が必要です。この情報は、Minishift または CDK の起動時に提供されます。

前提条件

- Fabric8 Launcher ツールがインストールされ、設定されている。

手順

1. Minishift または CDK を起動したコンソールに移動します。

2. 実行中の Fabric8 Launcher にアクセスするために使用できる URL およびユーザー認証情報のコンソール出力を確認します。

Minishift または CDK 起動からのコンソール出力の例

```
...
-- Removing temporary directory ... OK
-- Server Information ...
   OpenShift server started.
   The server is accessible via web console at:
      https://192.168.42.152:8443

   You are logged in as:
      User:     developer
      Password: developer

   To login as administrator:
      oc login -u system:admin
```

5.6.5.2. Fabric8 Launcher を使用したセキュア化されたサンプルアプリケーションの作成

前提条件

- 実行中の Fabric8 Launcher インスタンスの URL およびユーザー認証情報。詳細はを参照してください 「Fabric8 Launcher ツールの URL および認証情報の取得」。

手順

- ブラウザーで Fabric8 Launcher URL に移動します。
- 画面の指示に従って Node.js でサンプルを作成します。デプロイメントタイプを尋ねるプロンプトが表示されたら、I を選択するとローカルでビルドして実行します。
- 画面の指示に従います。

完了したら、Download as ZIP file ボタンをクリックし、ファイルをハードドライブに保存します。

5.6.5.3. oc CLI クライアントの認証

oc コマンドラインクライアントを使用して Minishift または CDK でアプリケーションの例を使用するには、Minishift または CDK Web インターフェースが提供するトークンを使用してクライアントを認証する必要があります。

前提条件

- 実行中の Fabric8 Launcher インスタンスの URL と Minishift または CDK のユーザー認証情報。詳細はを参照してください 「Fabric8 Launcher ツールの URL および認証情報の取得」。

手順

1. ブラウザーで Minishift または CDK URL に移動します。
2. Web コンソールの右上隅にある疑問符アイコンをクリックします。
3. ドロップダウンメニューで Command Line Tools を選択します。
4. oc login コマンドをコピーします。
5. 端末にコマンドを貼り付けます。このコマンドは、認証トークンを使用して Minishift または CDK アカウントで oc CLI クライアントを認証します。

```
$ oc login OPENSHIFT_URL --token=MYTOKEN
```

5.6.5.4. oc CLI クライアントを使用した Secured サンプルアプリケーションのデプロイ

本セクションでは、セキュアなサンプルアプリケーションをビルドし、コマンドラインから OpenShift にデプロイする方法を説明します。

前提条件

- Minishift または CDK で Fabric8 Launcher ツールを使用して作成されたアプリケーションのサンプル。詳細はを参照してください 「Fabric8 Launcher を使用したセキュア化されたサンプルアプリケーションの作成」。

- Fabric8 Launcher URL。

- 認証された oc クライアント。詳細はを参照してください 「oc CLI クライアントの認証」。

手順

1. GitHub からプロジェクトのクローンを作成します。

```
$ git clone git@github.com:USERNAME/MY_PROJECT_NAME.git
```

プロジェクトの ZIP ファイルをダウンロードした場合は、展開します。

```
$ unzip MY_PROJECT_NAME.zip
```

2. 新しい OpenShift プロジェクトを作成します。

```
$ oc new-project MY_PROJECT_NAME
```

3. アプリケーションのルートディレクトリーに移動します。

4. ZIP ファイルの例から service.sso.yaml ファイルを使用して、Red Hat SSO サーバーをデプロイします。

```
$ oc create -f service.sso.yaml
```

5. npm を使用して Minishift または CDK へのデプロイメントを開始します。

```
$ npm install && npm run openshift -- \
    -d SSO_AUTH_SERVER_URL=$(oc get route secure-sso -o jsonpath='{"https://"}{.spec.host}{"/auth\n"}')
```

これらのコマンドは、不足しているモジュール依存関係をすべてインストールし、Nodeshift モジュールを使用してサンプルアプリケーションを OpenShift にデプロイします。

5.6.6. セキュアなサンプルアプリケーションの OpenShift Container Platform へのデプロイ

Minishift または CDK のほかに、OpenShift Container Platform にサンプルを作成し、このサンプルを若干の違いでデプロイできます。最も重要な違いは、OpenShift Container Platform にデプロイする前に Minishift または CDK でサンプルアプリケーションを作成する必要があることです。

前提条件

- Minishift または CDK を使用して作成された例。

5.6.6.1. oc CLI クライアントの認証

oc コマンドラインクライアントを使用して OpenShift Container Platform でサンプルアプリケーションを使用するには、OpenShift Container Platform Web インターフェースによって提供されるトークンを使用してクライアントを認証する必要があります。

前提条件

- OpenShift Container Platform のアカウント。

手順

1. ブラウザーで OpenShift Container Platform URL に移動します。
2. Web コンソールの右上隅にある疑問符アイコンをクリックします。
3. ドロップダウンメニューで Command Line Tools を選択します。
4. oc login コマンドをコピーします。
5. 端末にコマンドを貼り付けます。このコマンドは、認証トークンを使用して oc CLI クライアントを OpenShift Container Platform アカウントで認証します。

```
$ oc login OPENSHIFT_URL --token=MYTOKEN
```

5.6.6.2. oc CLI クライアントを使用した Secured サンプルアプリケーションのデプロイ

本セクションでは、セキュアなサンプルアプリケーションをビルドし、コマンドラインから OpenShift にデプロイする方法を説明します。

前提条件

- Minishift または CDK で Fabric8 Launcher ツールを使用して作成されたアプリケーションのサンプル。
- 認証された oc クライアント。詳細はを参照してください 「oc CLI クライアントの認証」。

手順

1. GitHub からプロジェクトのクローンを作成します。

```
$ git clone git@github.com:USERNAME/MY_PROJECT_NAME.git
```

プロジェクトの ZIP ファイルをダウンロードした場合は、展開します。

```
$ unzip MY_PROJECT_NAME.zip
```

2. 新しい OpenShift プロジェクトを作成します。

```
$ oc new-project MY_PROJECT_NAME
```

3. アプリケーションのルートディレクトリーに移動します。

4. ZIP ファイルの例から service.sso.yaml ファイルを使用して、Red Hat SSO サーバーをデプロイします。

```
$ oc create -f service.sso.yaml
```

5. npm を使用して OpenShift Container Platform へのデプロイメントを開始します。

```
$ npm install && npm run openshift -- \
    -d SSO_AUTH_SERVER_URL=$(oc get route secure-sso -o jsonpath='{"https://"}
{.spec.host}{"/auth\n"}')
```

これらのコマンドは、不足しているモジュール依存関係をすべてインストールし、Nodeshift モジュールを使用してサンプルアプリケーションを OpenShift にデプロイします。

5.6.7. セキュアなアプリケーション API エンドポイントへの認証

Secured サンプルアプリケーションは、呼び出し元が認証され、承認された場合に GET リクエストを許可するデフォルトの HTTP エンドポイントを提供します。クライアントは最初に Red Hat SSO サーバーに対して認証を行い、認証ステップによって返されるアクセストークンを使用する Secured サンプルアプリケーションに対して GET リクエストを実行します。

5.6.7.1. セキュアなアプリケーション API エンドポイントの取得

クライアントを使用して例と対話する場合は、PROJECT_ID サービスである Secured サンプルアプリケーションエンドポイントを指定する必要があります。

前提条件

- セキュアなサンプルアプリケーションがデプロイされ、実行されている。
- 認証された oc クライアント。

手順

1. ターミナルアプリケーションで oc get routes コマンドを実行します。

   出力例を以下の表に示します。

例5.1 セキュアなエンドポイントの一覧

名前	ホスト/ポート	パス	サービス	ポート	termination

名前	ホスト/ポート	パス	サービス	ポート	termination
secure-sso	secure-sso-myproject.LOCAL_OPENSHIFT_HOSTNAME		secure-sso	<all>	passthrough
PROJECT_ID	PROJECT_ID-myproject.LOCAL_OPENSHIFT_HOSTNAME		PROJECT_ID	<all>	
SSO	sso-myproject.LOCAL_OPENSHIFT_HOSTNAME		SSO	<all>	

上記の例では、サンプルエンドポイントは http://PROJECT_ID-myproject.LOCAL_OPENSHIFT_HOSTNAME になり ます。PROJECT_ID は、developers.redhat.com/launch または Fabric8 Launcher ツールを使用してサンプルを生成する際に入力した名前に基づいています。

5.6.7.2. コマンドラインでの HTTP リクエストの認証

HTTP POST リクエストを Red Hat SSO サーバーに送信してトークンを要求します。以下の例では、CLI ツール jq を使用して JSON 応答からトークン値を抽出します。

前提条件

- セキュリティー保護されたエンドポイント URL のサンプル。詳細はを参照してください「セキュアなアプリケーション API エンドポイントの取得」。
- jq コマンドラインツール（任意）。ツールのダウンロードと、詳細な情報は、https://stedolan.github.io/jq/ を参照してください。

手順

1.

curl、認証情報、および <SSO_AUTH_SERVER_URL> でアクセストークンを要求し、jq コマンドで応答からトークンを抽出します。

```
curl -sk -X POST
https://<SSO_AUTH_SERVER_URL>/auth/realms/master/protocol/openid-
connect/token \
  -d grant_type=password \
  -d username=alice\
  -d password=password \
  -d client_id=demoapp \
  -d client_secret=1daa57a2-b60e-468b-a3ac-25bd2dc2eadc \
  | jq -r '.access_token'

eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICJRek1nbXhZMUhrQnpxTnR0Snk
wMm5jNTNtMGNiWDQxV1hNSTU1MFo4MGVBIn0.eyJqdGkiOiI0NDA3YTliNC04YWRhL
TRlMTctODQ2ZS03YjI5MjMyN2RmYTliLCJleHAiOjE1MDc3OTM3ODcsIm5iZiI6MCwiaW
F0IjoxNTA3NzkzNzI3LCJpc3MiOiJodHRwczovL3NlY3VyZS1zc28tc3NvLWRlbW8uYXBw
cy5jYWZlLWJhYmUub3JnL2F1dGgvcmVhbG1zL21hc3RlciIsImF1ZCI6ImRlbW9hcHAiL
CJzdWIiOiJjMDE3NWNjYi0wODkyLTRiMzEtODI5Zi1kZGE4NzM4MTVmZTgiLCJ0eXAiOi
JCZWFyZXIiLCJhenAiOiJkZW1vYXBwIiwiYXV0aF90aW1lIjowLCJzZXNzaW9uX3N0YXRl
IjoiMDFjOTkzNGQtNmZmOS00NWYzLWJkNWUtMTU4NDI5ZDZjNDczIiwiYWNyIjoiMSIsI
mNsaWVudF9zZXNzaW9uIjoiMzM3Yzk0MTYtYTdlZS00ZWUzLThjZWQtODhlODI0MGJjN
TAyIiwiYWxsb3dlZC1vcmlnaW5zIjpbIioiXSwicmVhbG1fYWNjZXNzIjp7InJvbGVzIjpbIm
vb3N0ZXItYWRtaW4iXX0sInJlc291cmNlX2FjY2VzcyI6eyJzZWN1cmVkLWJvb3N0ZXItZW
5kcG9pbnQiOnsicm9sZXMiOlsiYm9vc3Rlci1hZG1pbiJdfSwiYWNjb3VudCI6eyJyb2xlcyI
6WyJtYW5hZ2UtYWNjb3VudCIsInZpZXctcHJvZmlsZSJdfX0sIm5hbWUiOiJBbGljZSBJb
kNoYWlucyIsInByZWZlcnJlZF91c2VybmFtZSI6ImFsaWNlIiwiZ2l2ZW5fbmFtZSI6IkFsaW
NlIiwiZmFtaWx5X25hbWUiOiJJbkNoYWlucyIsImVtYWlsIjoiYWxpY2VAa2V5Y2xvYWwsub
3Jnln0.mjmZe37enHpigJv0BGulitOj-
kfMLPNwYzNd3n0Ax4Nga7KpnfytGyuPSvR4KAG8rzkfBNN9kIPYdy7pJEeYlfmnFUkM4
EDrZYgn4qZAznP1Wzy1RfVRdUFi0-
GqFTMPb37o5HRldZZ09QljX_j3GHnoMGXRtYW9RZN4eKkYkcz9hRwgfJoTy2CuwFqeJ
wZYUyXifrfA-JoTr0UmSUed-0NMksGrtJjjPggUGS-
qOn6OgKcmN2vaVAQlxW32y53JqUXctfLQ6DhJzlMYTmOfIlPy0sgG1mG7sovQhw1xTg
0vTjdx8zQ-EJcexkj7livRevRZsslKgqRFWs67jQAFQA
```

<SSO_AUTH_SERVER_URL> は secure-sso サービスの URL です。

ユーザー名、パスワード 、および client_secret などの属性は通常シークレット に保存されますが、上記のコマンドはこの例でデモ目的でこの認証情報を使用します。

jq を使用してトークンを抽出しない場合は、curl コマンドのみを実行して、アクセストークンを手動で抽出できます。

注記

-sk オプションは、自己署名証明書から生じる障害を無視するように curl に指示します。実稼働環境ではこのオプションを使用しないでください。macOS に、curl バージョン 7.56.1 以上がインストールされ て いる必要があります。OpenSSL で構築する必要があります。

1.

Secured サービスを呼び出します。アクセス（ベアラー）トークンを HTTP ヘッダーに割り当てます。

```
$ curl -v -H "Authorization: Bearer <TOKEN>" http://<SERVICE_HOST>/api/greeting

{
  "content": "Hello, World!",
  "id": 2
}
```

例5.2 Access(Bearer)トークンを含む GET リクエストヘッダーの例

```
> GET /api/greeting HTTP/1.1
> Host: <SERVICE_HOST>
> User-Agent: curl/7.51.0
> Accept: */*
> Authorization: Bearer <TOKEN>
```

<SERVICE_HOST> はセキュアなエンドポイントの URL です。詳細はを参照してください 「セキュアなアプリケーション API エンドポイントの取得」。

2.

アクセストークンの署名を確認します。

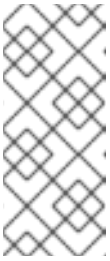
アクセストークンは JSON Web トークン であるため、JWT デバッガー を使用してデコードできます。

a.

Web ブラウザーで、JWT デバッガー の Web サイトに移動します。

b.

Algorithm ドロップダウンメニューから RS256 を選択します。



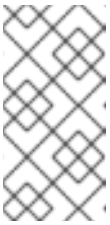
注記

選択後に Web フォームが更新され、正しい RSASHA256（...）セクションの情報。そうでない場合は、HS256 に切り替えてから RS256 に戻ります。

c.

VERIFY SIGNATURE セクションに、右端のテキストボックスに以下のコンテンツを貼り付け ます。

```
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAoETnPmN55xBJjRzN/cs
30OzJ9olkteLVNRjzdTxFOyRtS2ovDfzdhhO9XzUcTMbIsCOAZtSt8K+6yvBXypOSY
vl75EUdypmkcK1KoptqY5KEBQ1KwhWuP7IWQ0fshUwD6jI1QWDfGxfM/h34FvEn/0
tJ71xN2P8TI2YanwuDZgosdobx/PAvIGREBGuk4BgmexTOkAdnFxIUQcCkiEZ2C41
uCrxiS4CEe5OX91aK9HKZV4ZJX6vnqMHmdDnsMdO+UFtxOBYZio+a1jP4W3d7J5f
GeiOaXjQCOpivKnP2yU2DPdWmDMyVb67l8DRA+jh0OJFKZ5H2fNgE3Il59vdsRwID
AQAB
-----END PUBLIC KEY-----
```



注記

これは、Secured サンプルアプリケーションの Red Hat SSO サーバーデプロイメントからのマスターレルムの公開鍵です。

d.

トークン 出力をクライアント出力から Encodedcoded ボックスに貼り 付け ます。

Signature Verified sign が Debug ページに表示されます。

5.6.7.3. Web インターフェースを使用した HTTP 要求の認証

セキュリティー保護されたエンドポイントには、HTTP API の他に、対話する Web インターフェースも含まれます。

以下の手順は、セキュリティーの適用方法、認証方法、および認証トークンの使用方法を確認することです。

前提条件

- セキュリティーが保護されたエンドポイント URL。詳細はを参照してください 「セキュアなアプリケーション API エンドポイントの取得」。

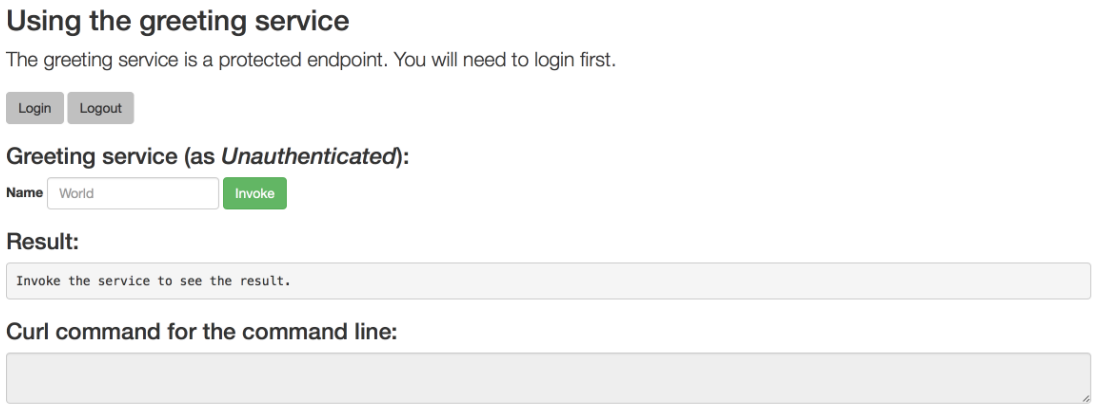
手順

1. Web ブラウザーで、エンドポイント URL に移動します。

2. 認証されていない要求を実行します。

   a. Invoke ボタンをクリックします。

図5.1 認証されていないセキュアな Web インターフェースの例



サービスは、HTTP 403 Forbidden ステータスコードで応答します。

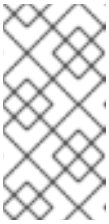
注記

これは正しいステータスコードではありません。HTTP 401 Unauthorized である必要があり ます。この問題 は特定され、この例は解決されるとすぐに更新されます。

3. ユーザーとして認証された要求を実行します。

   a. ログイン ボタンをクリックして Red Hat SSO に対して認証します。SSO サーバーにリダイレクトされます。

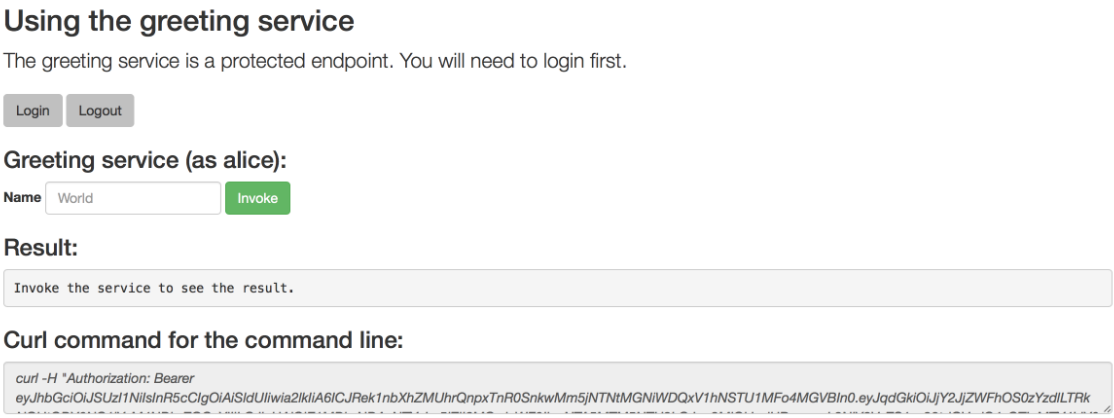
   b. WebSphere ユーザーとして ログイン し ます。Web インターフェースにリダイレクトされます。



注記

アクセス（ベアラー）のトークンは、ページの下部にあるコマンドライン出力に表示されます。

図5.2 認証されたセキュア化されたサンプル Web インターフェース（バリアントとしての）

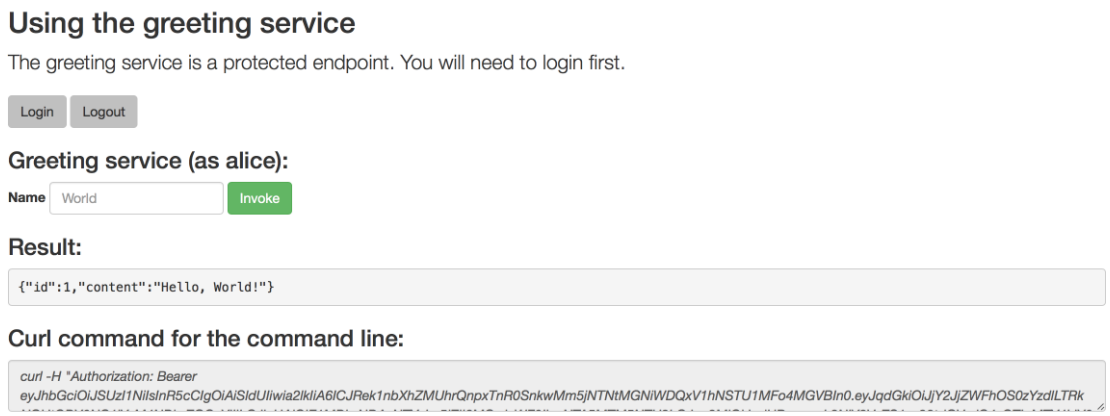


c.

Invoke を再びクリックし、Greeting サービスにアクセスします。

例外がなく、JSON 応答ペイロードが表示されることを確認します。これは、サービスがアクセス（ベアラー）トークンを受け入れ、Greeting サービスへのアクセスが承認されることを意味します。

図5.3 認証された Greeting Request の結果



d.

ログアウトします。

4.

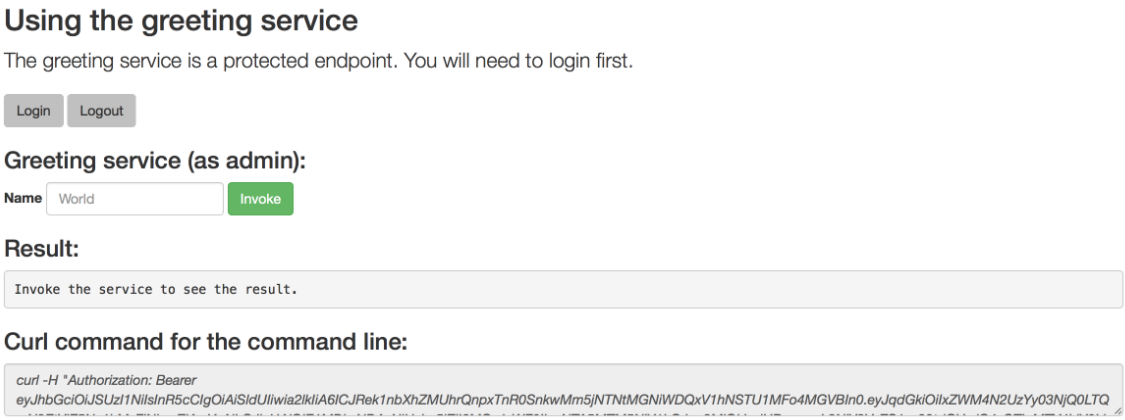
認証された要求を管理者として実行します。

a. **Invoke ボタンをクリックします。**

認証されていないリクエストを Greeting サービスに送信することを確認します。

b. **ログイン ボタンをクリックして、管理ユーザー として ログイン し ます。**

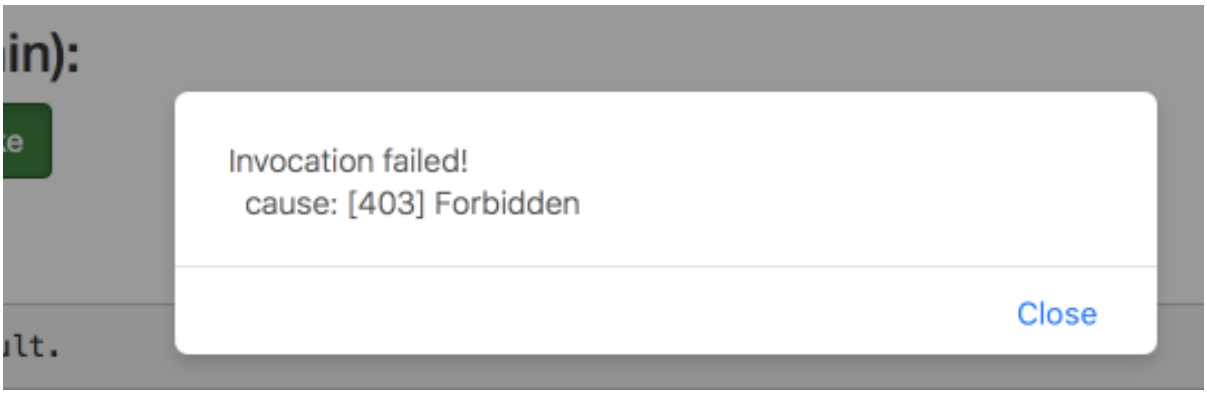
図5.4 認証されたセキュアな Web インターフェース（admin として）



5. **Invoke ボタンをクリックします。**

admin ユーザーは Greeting サービスへのアクセスが許可されていないため、このサービスは HTTP 403 Forbidden ステータスコードで応答します。

図5.5 承認されていません



5.6.8. セキュリティー保護された SSO リソース

OAuth2 仕様の背後にある原則に関する追加情報と、Red Hat SSO および Keycloak を使用してアプリケーションのセキュリティー保護を行うには、以下のリンクに従ってください。

- Aaron Parecki: OAuth2 簡体字
- Red Hat SSO 7.1 ドキュメント
- Keycloak 3.2 Documentation
- Spring Boot にセキュア化
- Eclipse Vert.x でセキュア化
- Thorntail に対してセキュア化

5.7. NODE.JS のキャッシュ例

重要

以下の例は、実稼働環境で実行することは意図されていません。

制限： Minishift または CDK でこのサンプルアプリケーションを実行します。手動でのワークフローを使用して、この例を OpenShift Online Pro および OpenShift Container Platform にデプロイすることもできます。この例では、現在 OpenShift Online ドキュメンテーションでは使用できません。

頻度レベルの例： 高度。

Cache の例では、キャッシュを使用してアプリケーションの応答時間を向上させる方法を例示します。

この例では、以下の方法を紹介します。

- キャッシュを OpenShift にデプロイします。

- アプリケーション内でキャッシュを使用します。

5.7.1. キャッシュの仕組みと必要な場合にキャッシュがどのように機能するか

キャッシュを使用すると、情報を保存し、一定期間アクセスできます。元のサービスを繰り返し呼び出すよりも、キャッシュ内の情報に問題なくアクセスすることができます。キャッシュを使用する欠点は、キャッシュされた情報が最新の状態ではないことです。ただし、キャッシュに保存されている各値に expiration または TTL(time to live)を設定することで、この問題を軽減することができます。

> **例5.3 キャッシュの例**
>
> service 1 と service 2 の 2 つのアプリケーションがあると します。
>
> - service1 は、service2 からの値によって異なります。
>   - service2 からの値が頻繁に変更されると、service1 は一定期間 service2 から値をキャッシュする可能性があります。
>   - キャッシュされた値を使用すると、service2 が呼び出される回数を減らすこともできます。
> - service1 500 ミリ秒を取得して service2 から直接値を取得し、キャッシュ値を取得する 100 ミリ秒の場合、service1 はキャッシュされた各呼び出しに対してキャッシュされた値を使用して 400 ミリ秒を保存します。
> - service1 が、1 秒あたり service2 5 へのキャッシュされていない呼び出しを行い、10 秒を超える呼び出しを行う場合、これは 50 回の呼び出しになります。
> - service1 が、代わりに TTL が 1 秒のキャッシュされた値を使用して開始されると、10 秒間の呼び出し数が減ります。

Cache サンプルの仕組み

1. キャッシュ、カット 名、 および greeting サービスがデプロイされ、公開されます。

2. ユーザーは greeting サービスの web フロントエンドにアクセスします。

3. ユーザーは、Web フロントエンドでボタンを使用して greeting HTTP API を呼び出します。

4. greeting サービスは、カット 名 サービスからの値によって異なります。

   - greeting サービスは、最初にその値が キャッシュ サービスに保存されているかどうかを確認します。設定されている場合、キャッシュされた値が返されます。

   - 値がキャッシュされていない場合、greeting サービスはカット 名 サービスを呼び出して値を返し、TTL を 5 秒で キャッシュ サービスに保存します。

5. Web フロントエンドには、greeting サービスからの応答と、操作の合計時間が表示されます。

6. ユーザーはサービスを複数回呼び出し、キャッシュされた操作とキャッシュされていない操作の違いを確認します。

   - キャッシュ操作はキャッシュされていない操作よりもはるかに高速です。

   - ユーザーは、TTL の期限が切れる前にキャッシュを強制的に消去できます。

5.7.2. Cache サンプルアプリケーションの OpenShift Online へのデプロイ

以下のオプションのいずれかを使用して、OpenShift Online で Cache サンプルアプリケーションを実行します。

- developers.redhat.com/launch の使用

- oc CLI クライアントの使用

各方法は同じ oc コマンドを使用してアプリケーションをデプロイしますが、developers.redhat.com/launch を使用すると、oc コマンドを実行する自動デプロイメントワークフローが提供されます。

5.7.2.1. developers.redhat.com/launch を使用したサンプルアプリケーションのデプロイ

このセクションでは、REST API Level 0 のサンプルアプリケーションをビルドし、Red Hat Developer Launcher Web インターフェースから OpenShift にデプロイする方法を説明します。

前提条件

- OpenShift Online のアカウント。

手順

1. ブラウザーで developers.redhat.com/launch URL に移動します。
2. 画面の指示に従って、Node.js でサンプルアプリケーションを作成して起動します。

5.7.2.2. oc CLI クライアントの認証

oc コマンドラインクライアントを使用して OpenShift Online でアプリケーションのサンプルを使用するには、OpenShift Online Web インターフェースによって提供されるトークンを使用してクライアントを認証する必要があります。

前提条件

- OpenShift Online のアカウント。

手順

1. ブラウザーで OpenShift Online URL に移動します。
2. Web コンソールの右上隅にある疑問符アイコンをクリックします。
3. ドロップダウンメニューで Command Line Tools を選択します。

4. oc login コマンドをコピーします。

5. 端末にコマンドを貼り付けます。このコマンドは、認証トークンを使用して CLI クライアント oc を OpenShift Online アカウントで認証します。

```
$ oc login OPENSHIFT_URL --token=MYTOKEN
```

5.7.2.3. oc CLI クライアントを使用した Cache サンプルアプリケーションのデプロイ

本セクションでは、コマンドラインから Cache サンプルアプリケーションをビルドし、これを OpenShift にデプロイする方法を説明します。

前提条件

- developers.redhat.com/launch を使用して作成されたサンプルアプリケーション。詳細はを参照してください 「developers.redhat.com/launch を使用したサンプルアプリケーションのデプロイ」。
- 認証された oc クライアント。詳細はを参照してください 「oc CLI クライアントの認証」。

手順

1. GitHub からプロジェクトのクローンを作成します。

```
$ git clone git@github.com:USERNAME/MY_PROJECT_NAME.git
```

プロジェクトの ZIP ファイルをダウンロードした場合は、展開します。

```
$ unzip MY_PROJECT_NAME.zip
```

2. 新規プロジェクトを作成します。

```
$ oc new-project MY_PROJECT_NAME
```

3. アプリケーションのルートディレクトリーに移動します。

4. キャッシュサービスをデプロイします。

```
$ oc apply -f service.cache.yml
```

注記

x86_64 以外のアーキテクチャーを使用している場合は、YAML ファイルで Red Hat Data Grid のイメージ名をそのアーキテクチャーの関連するイメージ名に更新します。たとえば、s390x アーキテクチャーの場合は、イメージ名を IBM Z イメージ名 registry.access.redhat.com/jboss-datagrid-7/datagrid73-openj9-11-openshift-rhel8 に更新します。

5. start-openshift.sh を使用して OpenShift へのデプロイメントを開始します。

```
$ ./start-openshift.sh
```

6. アプリケーションのステータスを確認し、Pod が実行されていることを確認します。

```
$ oc get pods -w
NAME                        READY     STATUS      RESTARTS   AGE
cache-server-123456789-aaaaa            1/1       Running     0          8m
MY_APP_NAME-cutename-1-bbbbb       1/1       Running     0          4m
MY_APP_NAME-cutename-s2i-1-build   0/1       Completed   0          7m
MY_APP_NAME-greeting-1-ccccc       1/1       Running     0          3m
MY_APP_NAME-greeting-s2i-1-build   0/1       Completed   0          3m
```

3 つの Pod が完全にデプロイされ、起動されると、ステータスが Running である必要があります。

7. サンプルアプリケーションがデプロイされ、起動したら、そのルートを決定します。

ルート情報の例

```
$ oc get routes
NAME              HOST/PORT                                       PATH      SERVICES
PORT      TERMINATION
MY_APP_NAME-cutename   MY_APP_NAME-cutename-
MY_PROJECT_NAME.OPENSHIFT_HOSTNAME            MY_APP_NAME-cutename
8080                 None
```

```
MY_APP_NAME-greeting   MY_APP_NAME-greeting-
MY_PROJECT_NAME.OPENSHIFT_HOSTNAME          MY_APP_NAME-greeting   8080
None
```

Pod のルート情報は、アクセスに使用するベース URL を提供します。上記の例では、http://MY_APP_NAME-greeting-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME をベース URL として使用して greeting サービスにアクセスします。

5.7.3. Cache サンプルアプリケーションの Minishift または CDK へのデプロイ

以下のオプションのいずれかを使用して、Minishift または CDK で Cache サンプルアプリケーションをローカルに実行します。

- Fabric8 Launcher の使用
- oc CLI クライアントの使用

各方法は同じ oc コマンドを使用してアプリケーションをデプロイしますが、Fabric8 Launcher を使用すると oc コマンドを実行する自動デプロイメントワークフローが提供されます。

5.7.3.1. Fabric8 Launcher ツールの URL および認証情報の取得

Minishift または CDK でサンプルアプリケーションを作成してデプロイするには、Fabric8 Launcher ツール URL およびユーザー認証情報が必要です。この情報は、Minishift または CDK の起動時に提供されます。

前提条件

- Fabric8 Launcher ツールがインストールされ、設定されている。

手順

1. Minishift または CDK を起動したコンソールに移動します。
2.

実行中の Fabric8 Launcher にアクセスするために使用できる URL およびユーザー認証情報のコンソール出力を確認します。

Minishift または CDK 起動からのコンソール出力の例

```
...
-- Removing temporary directory ... OK
-- Server Information ...
   OpenShift server started.
   The server is accessible via web console at:
      https://192.168.42.152:8443

   You are logged in as:
      User:     developer
      Password: developer

   To login as administrator:
      oc login -u system:admin
```

5.7.3.2. Fabric8 Launcher ツールを使用したサンプルアプリケーションのデプロイ

本セクションでは、REST API レベル 0 サンプルアプリケーションをビルドし、それを Fabric8 Launcher Web インターフェースから OpenShift にデプロイする方法を説明します。

前提条件

- 実行中の Fabric8 Launcher インスタンスの URL と Minishift または CDK のユーザー認証情報。詳細はを参照してください 「Fabric8 Launcher ツールの URL および認証情報の取得」。

手順

1. ブラウザーで Fabric8 Launcher URL に移動します。
2. 画面の指示に従って、Node.js でサンプルアプリケーションを作成して起動します。

5.7.3.3. oc CLI クライアントの認証

oc コマンドラインクライアントを使用して Minishift または CDK でアプリケーションの例を使用するには、Minishift または CDK Web インターフェースが提供するトークンを使用してクライアントを認証する必要があります。

前提条件

- 実行中の Fabric8 Launcher インスタンスの URL と Minishift または CDK のユーザー認証情報。詳細はを参照してください 「Fabric8 Launcher ツールの URL および認証情報の取得」。

手順

1. ブラウザーで Minishift または CDK URL に移動します。
2. Web コンソールの右上隅にある疑問符アイコンをクリックします。
3. ドロップダウンメニューで Command Line Tools を選択します。
4. oc login コマンドをコピーします。
5. 端末にコマンドを貼り付けます。このコマンドは、認証トークンを使用して Minishift または CDK アカウントで oc CLI クライアントを認証します。

```
$ oc login OPENSHIFT_URL --token=MYTOKEN
```

5.7.3.4. oc CLI クライアントを使用した Cache サンプルアプリケーションのデプロイ

本セクションでは、コマンドラインから Cache サンプルアプリケーションをビルドし、これを OpenShift にデプロイする方法を説明します。

前提条件

- Minishift または CDK で Fabric8 Launcher ツールを使用して作成されたアプリケーションのサンプル。詳細はを参照してください 「Fabric8 Launcher ツールを使用したサンプルアプリケーションのデプロイ」。

- Fabric8 Launcher ツール URL。
- 認証された oc クライアント。詳細はを参照してください 「oc CLI クライアントの認証」。

手順

1. GitHub からプロジェクトのクローンを作成します。

```
$ git clone git@github.com:USERNAME/MY_PROJECT_NAME.git
```

プロジェクトの ZIP ファイルをダウンロードした場合は、展開します。

```
$ unzip MY_PROJECT_NAME.zip
```

2. 新規プロジェクトを作成します。

```
$ oc new-project MY_PROJECT_NAME
```

3. アプリケーションのルートディレクトリーに移動します。

4. キャッシュサービスをデプロイします。

```
$ oc apply -f service.cache.yml
```

注記

x86_64 以外のアーキテクチャーを使用している場合は、YAML ファイルで Red Hat Data Grid のイメージ名をそのアーキテクチャーの関連するイメージ名に更新します。たとえば、s390x アーキテクチャーの場合は、イメージ名を IBM Z イメージ名 registry.access.redhat.com/jboss-datagrid-7/datagrid73-openj9-11-openshift-rhel8 に更新します。

5. start-openshift.sh を使用して OpenShift へのデプロイメントを開始します。

```
$ ./start-openshift.sh
```

6. アプリケーションのステータスを確認し、Pod が実行されていることを確認します。

```
$ oc get pods -w
NAME                          READY     STATUS      RESTARTS   AGE
cache-server-123456789-aaaaa            1/1       Running     0          8m
MY_APP_NAME-cutename-1-bbbbb       1/1       Running     0          4m
MY_APP_NAME-cutename-s2i-1-build   0/1       Completed   0          7m
MY_APP_NAME-greeting-1-ccccc       1/1       Running     0          3m
MY_APP_NAME-greeting-s2i-1-build   0/1       Completed   0          3m
```

3 つの Pod が完全にデプロイされ、起動されると、ステータスが Running である必要があります。

7. サンプルアプリケーションがデプロイされ、起動したら、そのルートを決定します。

ルート情報の例

```
$ oc get routes
NAME              HOST/PORT                                       PATH      SERVICES
PORT      TERMINATION
MY_APP_NAME-cutename   MY_APP_NAME-cutename-
MY_PROJECT_NAME.OPENSHIFT_HOSTNAME          MY_APP_NAME-cutename
8080                None
MY_APP_NAME-greeting   MY_APP_NAME-greeting-
MY_PROJECT_NAME.OPENSHIFT_HOSTNAME          MY_APP_NAME-greeting   8080
None
```

Pod のルート情報は、アクセスに使用するベース URL を提供します。上記の例では、http://MY_APP_NAME-greeting-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME をベース URL として使用して greeting サービスにアクセスします。

5.7.4. Cache サンプルアプリケーションの OpenShift Container Platform へのデプロイ

サンプルアプリケーションを OpenShift Container Platform に作成し、デプロイするプロセスは OpenShift Online と似ています。

前提条件

- developers.redhat.com/launch を使用して作成されたサンプルアプリケーション。

手順

- の手順に従って 「Cache サンプルアプリケーションの OpenShift Online へのデプロイ」、OpenShift Container Platform Web コンソールからの URL およびユーザー認証情報のみを使用します。

5.7.5. 変更されていないキャッシュサンプルアプリケーションとの対話

デフォルトの Web インターフェースを使用して変更されていないキャッシュサンプルアプリケーションと対話し、頻繁にアクセスされるデータを保存すると、サービスへのアクセスに必要な時間が短くなる可能性があることに注意してください。

前提条件

- アプリケーションがデプロイされている必要があります。

手順

1. ブラウザーを使用して greeting サービスに移動します。
2. サービスを 1 度、Invoke the service を クリックします。

   duration の値は 2000 を超えることに注意してください。また、キャッシュの状態が No cached 値から A 値キャッシュ に変更されていることに注意してください。
3. 5 秒待機し、キャッシュ状態が No cached 値 に戻されました。

   キャッシュされた値の TTL は 5 秒に設定されます。TTL の期限が切れると、値はキャッシュされなくなります。
4. サービスを 1 度再度クリックして、値をキャッシュします。
5.

キャッシュの状態 が A 値がキャッシュされている間に、サービスの呼び出し を数秒で繰り返し実行し ます。

キャッシュされた値を使用するため、期間 の値が大幅に低くなることに注意してください。キャッシュのクリアをクリックすると、キャッシュ のプリエンプションが実行されます。

5.7.6. キャッシュリソース

キャッシュの背景や関連情報は、以下を参照してください。

- Spring Boot のキャッシュ
- Eclipse Vert.x のキャッシュ
- Thorntail のキャッシュ

付録A NODESHIFT について

Nodeshift は、Node.js プロジェクトを使用して OpenShift デプロイメントを実行するモジュールです。

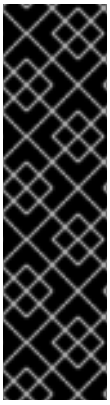
重要

Nodeshift は oc CLI クライアントがインストールされ、OpenShift クラスターにログインしていることを前提としています。Nodeshift は、oc CLI クライアントが使用している現在のプロジェクトも使用します。

Nodeshift は、プロジェクトのルートにある .nodeshift フォルダーにあるリソースファイルを使用して、OpenShift Routes、サービス、および DeploymentConfig の作成を処理します。Nodeshift の詳細は、Nodeshift プロジェクトページを参照してください。

付録B アプリケーションのデプロイメント設定の更新

サンプルアプリケーションのデプロイメント設定には、ルート情報や readiness プローブの場所など、OpenShift でのアプリケーションのデプロイおよび実行に関連する情報が含まれます。サンプルアプリケーションのデプロイメント設定は YAML ファイルのセットに保存されます。Fabric8 Maven プラグインを使用する例の場合、YAML ファイルは src/main/fabric8/ ディレクトリーにあります。Nodeshift を使用する例の場合、YAML ファイルは .nodeshift ディレクトリーにあり ます。



重要

Fabric8 Maven Plugin および Nodeshift によって使用されるデプロイメント設定ファイルは、完全な OpenShift リソース定義である必要はありません。Fabric8 Maven Plugin と Nodeshift の両方は、デプロイメント設定ファイルを取り、欠落している情報を追加して完全な OpenShift リソース定義を作成できます。Fabric8 Maven プラグインによって生成されたリソース定義は target/classes/META-INF/fabric8/ ディレクトリーにあります。Nodeshift によって生成されたリソース定義は tmp/nodeshift/resource/ ディレクトリーにあります。

前提条件

- 既存のサンプルプロジェクト。
- oc CLI クライアントがインストールされていること。

手順

1. 既存の YAML ファイルを編集するか、または設定更新で追加の YAML ファイルを作成します。
   - たとえば、この例に readinessProbe が設定された YAML ファイルがある場合、path の値を利用可能な異なる パス に変更して、readiness の有無を確認できます。

```
spec:
  template:
    spec:
      containers:
        readinessProbe:
          httpGet:
            path: /path/to/probe
            port: 8080
            scheme: HTTP
...
```

- readinessProbe が既存の YAML ファイルに設定されていない場合、readinessProbe 設定で同じディレクトリーに新規 YAML ファイルを作成することもできます。

2. Maven または npm を使用して、例の更新バージョンをデプロイします。

3. 設定の更新が、デプロイしたバージョンの例に表示されることを確認します。

```
$ oc export all --as-template='my-template'

apiVersion: template.openshift.io/v1
kind: Template
metadata:
  creationTimestamp: null
  name: my-template
objects:
- apiVersion: template.openshift.io/v1
  kind: DeploymentConfig
  ...
  spec:
    ...
    template:
      ...
      spec:
        containers:
          ...
          livenessProbe:
            failureThreshold: 3
            httpGet:
              path: /path/to/different/probe
              port: 8080
              scheme: HTTP
            initialDelaySeconds: 60
            periodSeconds: 30
            successThreshold: 1
            timeoutSeconds: 1
          ...
```

関連情報

Web ベースのコンソールまたは oc CLI クライアントを使用してアプリケーションの設定を直接更新した場合、これらの変更を YAML ファイルにエクスポートし、追加します。oc export all コマンドを使用して、デプロイされたアプリケーションの設定を表示します。

付録C NODESHIFT を使用して NODE.JS アプリケーションをデプロイする JENKINS のフリースタイルプロジェクトの設定

ローカルホストから nodeshift を使用して Node.js アプリケーションをデプロイするのと同様に、Jenkins が nodeshift を使用して Node.js アプリケーションをデプロイするように設定できます。

前提条件

- OpenShift クラスターへのアクセス
- 同じ OpenShift クラスターで実行している Jenkins コンテナーイメージ。
- Jenkins サーバーにインストールされているNode.js プラグイン。
- nodeshift および Red Hat ベースイメージを使用するように設定された Node.js アプリケーション。

  nodeshift での Red Hat ベースイメージの使用例

  ```
  $ nodeshift --dockerImage=registry.access.redhat.com/rhscl/ubi8/nodejs-12 ...
  ```

- GitHub で利用可能なアプリケーションのソース。

手順

1. アプリケーションの新しい OpenShift プロジェクトを作成します。
   a. OpenShift Web コンソールを開き、ログインします。
   b. Create Project をクリックし、新しい OpenShift プロジェクトを作成します。

c. プロジェクト情報を入力して、Create をクリックします。

2. Jenkins がそのプロジェクトにアクセスできることを確認します。

たとえば、Jenkins のサービスアカウントを設定している場合は、アカウントのアプリケーションのプロジェクトへの 編集 アクセスがあることを確認します。

3. Jenkins サーバーで新しい フリースタイルの Jenkins プロジェクト を作成します。

a. New Item をクリックし ます。

b. 名前を入力して Freestyle project を選択し、OK をクリックし ます。

c. Source Code Management で Git を選択し、アプリケーションの GitHub URL を追加します。

d. Build Environment で、Provide Node & npm bin/ フォルダーを PATH にチェックし、Node.js 環境が設定されていることを確認してください。

e. Build で Add build step を 選択し、Execute Shell を選択します。

f. 以下を Command エリアに追加します。

```
npm install -g nodeshift
nodeshift --dockerImage=registry.access.redhat.com/rhscl/ubi8/nodejs-12 --
namespace=MY_PROJECT
```

MY_PROJECT を、アプリケーションの OpenShift プロジェクトの名前に置き換えます。

g. Save をクリックし ます。

4. Jenkins プロジェクトのメインページから Build Now をクリックし、アプリケーションが

ビルドされ、アプリケーションの OpenShift プロジェクトにデプロイします。

また、アプリケーションの OpenShift プロジェクトでルートを開いて、アプリケーションがデプロイされていることを確認することもできます。

次のステップ

- GITSCM ポーリング を追加すること、または the Poll SCM ビルドトリガー を使用することを検討してください。これらのオプションにより、新規コミットが GitHub リポジトリーにプッシュされるたびにビルドを実行できます。

- Node.js プラグインを設定する際に、グローバルパッケージとして nodeshift を 追加することを検討してください。これにより、Execute Shell ビルドステップを追加する 際に npm install -g nodeshift を省略できます。

- デプロイ前にテストを実行するビルドステップを追加することを検討してください。

付録D PACKAGE.JSON プロパティーの内訳

nodejs-rest-http/package.json

```
{
  "name": "nodejs-rest-http",
  "version": "1.1.1",
  "author": "Red Hat, Inc.",
  "license": "Apache-2.0",
  "scripts": {
    "test": "tape test/*.js | tap-spec", 1
    "lint": "eslint test/*.js app.js bin/*",
    "prepare": "nsp check",
    "coverage": "nyc npm test",
    "coveralls": "nyc npm test && nyc report --reporter=text-lcov | coveralls",
    "ci": "npm run lint && npm run coveralls",
    "dependencyCheck": "szero . --ci",
    "release": "standard-version",
    "openshift": "nodeshift --strictSSL=false --nodeVersion=8.x", 2
    "postinstall": "license-reporter report && license-reporter save --xml licenses.xml",
    "start": "node ." 3
  },
  "main": "./bin/www", 4
  "repository": {
    "type": "git",
    "url": "git://github.com/nodeshift-starters/nodejs-rest-http.git"
  },
  "files": [ 5
    "package.json",
    "app.js",
    "public",
    "bin",
    "LICENSE",
    "licenses"
  ],
  "bugs": {
    "url": "https://github.com/nodeshift-starters/nodejs-rest-http/issues"
  },
  "homepage": "https://github.com/nodeshift-starters/nodejs-rest-http",
  "devDependencies": { 6
    "coveralls": "^3.0.0",
    "nodeshift": "^1.3.0",
    "nsp": "~3.1.0",
    "nyc": "~11.4.1",
    "standard-version": "^4.2.0",
    "supertest": "^3.0.0",
    "szero": "^1.0.0",
    "tap-spec": "~4.1.1",
    "tape": "~4.8.0",
    "xo": "~0.20.3"
  },
```

```
  "dependencies": { 7
    "body-parser": "^1.18.2",
    "debug": "^3.1.0",
    "express": "^4.16.0",
    "license-reporter": "^1.1.3"
  }
}
```

1

ユニットテストを実行する npm スクリプト。npm run test で実行し ます。

2

このアプリケーションを Minishift または CDK にデプロイする npm スクリプト。npm run openshift で実行さ れます。strictSSL オプション を使用すると、自己署名証明書で Minishift または CDK インスタンスにデプロイできます。

3

このアプリケーションを起動する npm スクリプト。npm start で実行されます。

4

npm start で実行する際のアプリケーションの主なエントリーポイント。

5

Minishift または CDK にアップロードされるバイナリーに含まれるファイルを指定します。

6

npm レジストリーからインストールされる開発依存関係の一覧。これらは、Minishift または CDK のテストおよびデプロイメントに使用されます。

7

npm レジストリーからインストールされる依存関係の一覧。

付録E 追加の NODE.JS リソース

- Node.js Home Page
- npm Home Page

付録F アプリケーション開発リソース

OpenShift を使用したアプリケーションの開発に関する詳細は、以下を参照してください。

- OpenShift インタラクティブラーニングポータル（英語）

ネットワーク負荷を短縮し、アプリケーションのビルド時間を短縮するには、Minishift または CDK で Maven の Nexus ミラーを設定します。

- Maven 用の Nexus ミラーリングの設定

付録G SOURCE-TO-IMAGE(S2I)ビルドプロセス

Source-to-Image (S2I) は、アプリケーションソースのあるオンライン SCM リポジトリーから再現可能な Docker 形式のコンテナーイメージを生成するビルドツールです。S2I ビルドを使用すると、ビルド時間が短縮され、リソースおよびネットワークの使用の短縮、セキュリティーの改善、およびその他の利点とともに、最新バージョンのアプリケーションを簡単に配信できます。OpenShift は、複数のビルドストラテジーおよび入力ソース をサポートします。

詳細は、OpenShift Container Platform ドキュメントの「Source-to-Image (S2I) ビルド」の章を参照してください。

最終的なコンテナーイメージをアセンブルするには、S2I プロセスに 3 つの要素を指定する必要があります。

- GitHub などのオンライン SCM リポジトリーでホストされるアプリケーションソース。
- S2I Builder イメージは、アセンブルされたイメージの基盤として機能し、アプリケーションが実行されるエコシステムを提供します。
- 必要に応じて、S2I スクリプト によって使用される環境変数およびパラメーターを指定することもできます。

プロセスは、S2I スクリプトで指定された手順に従って、アプリケーションソースと依存関係をビルダーイメージに注入し、アセンブルされたアプリケーションを実行する Docker 形式のコンテナーイメージを生成します。詳細は、OpenShift Container Platform ドキュメントの 「S2I build requirements」、「build options」、および 「how builds work」を参照してください。

付録H 緊急レベル

利用可能な各例では、特定の最小知識が必要な概念を紹介しています。この要件は例によって異なります。最小要件と概念は、いくつかのレベルで組織化されています。ここで説明するレベルのほかに、各例に固有の追加情報が必要になる場合があります。

基本的

通常、Foundational Proficiency（英語版）は、サブジェクトに関する事前知識は必要ありません。また、重要な要素、概念、用語の一般的な認識とデモを提供します。例の説明にしたがって直接記載されているものを除き、特別な要件はありません。

Advanced

高度な例を使用する場合には、Kubernetes および OpenShift に加えて、サンプルのサブジェクトエリアの一般的な概念と用語に精通していることを前提としています。また、サービスとアプリケーションの設定やネットワークの管理など、独自に基本的なタスクを実行する必要もあります。この例でサービスが必要であるにもかかわらず、例の範囲外にある場合は、適切に設定する権限を持ち、サービスの状態のみがドキュメントに記載されています。

エキスパート

専門的な例としては、このサブジェクトに関する最高の知識が必要です。機能ベースのドキュメントやマニュアルに基づくタスクが多数あることが予想されており、ドキュメントは最も複雑なシナリオを対象としています。

付録I 用語集

I.1. 製品およびプロジェクト名

Developer Launcher(developers.redhat.com/launch)

Developer Launcher (developers.redhat.com/launch) は、Red Hat が提供するスタンドアロンの入門エクスペリエンスです。これは、OpenShift でクラウドネイティブな開発を開始するのに役立ちます。これには、OpenShift にダウンロード、ビルド、デプロイできる機能サンプルアプリケーションが含まれます。

minishift または CDK

Minishift を使用してマシンで実行されている OpenShift クラスター。

I.2. DEVELOPER LAUNCHER に固有の用語

例

REST API のある Web サービスなど、アプリケーション仕様。

通常、実行すべき言語やプラットフォームは指定しないでください。説明には目的の機能のみが含まれます。

アプリケーションの例

特定の ランタイム における特定 サンプル の言語固有の実装。アプリケーションのサンプルは、サンプルカタログ に一覧表示されます。

たとえば、アプリケーションの例は、Thorntail ランタイムを使用して実装された REST API を備えた Web サービスです。

サンプルカタログ

アプリケーションの例に関する情報が含まれる Git リポジトリー。

ランタイム

サンプルアプリケーションを実行する プラットフォーム。たとえば、Thorntail または Eclipse Vert.x などです。