Red Hat JBoss Enterprise Application Platform 8.0

OpenShift Container Platform 上の JBoss EAP の使用

Red Hat JBoss Enterprise Application Platform for OpenShift での開発ガイド

Last Updated: 2024-03-28

Red Hat JBoss Enterprise Application Platform 8.0 OpenShift Container Platform 上の JBoss EAP の使用

Red Hat JBoss Enterprise Application Platform for OpenShift での開発ガイド

法律上の通知

概要

Red Hat JBoss Enterprise Application Platform for OpenShift の使用ガイド

目次

JBOSS EAP ドキュメントへのフィードバック (英語のみ)

エラーを報告したり、ドキュメントを改善したりするには、Red Hat Jira アカウントにログインし、課題を送信してください。Red Hat Jira アカウントをお持ちでない場合は、アカウントを作成するように求められます。

手順

1. **このリンクをクリック** してチケットを作成します。
2. **Summary** に課題の簡単な説明を入力します。
3. **Description** に課題や機能拡張の詳細な説明を入力します。問題があるドキュメントのセクションへの URL を含めてください。
4. **Submit** をクリックすると、課題が作成され、適切なドキュメントチームに転送されます。

多様性を受け入れるオープンソースの強化

Red Hat では、コード、ドキュメント、Web プロパティーにおける配慮に欠ける用語の置き換えに取り組んでいます。まずは、マスター (master)、スレーブ (slave)、ブラックリスト (blacklist)、ホワイトリスト (whitelist) の 4 つの用語の置き換えから始めます。この取り組みは膨大な作業を要するため、今後の複数のリリースで段階的に用語の置き換えを実施して参ります。詳細は、Red Hat CTO である Chris Wright のメッセージ をご覧ください。

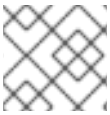
第1章 RED HAT JBOSS ENTERPRISE APPLICATION PLATFORM とは

Red Hat JBoss Enterprise Application Platform 8.0 (JBoss EAP) は、オープン標準に基づいて構築されたミドルウェアプラットフォームで、Jakarta EE 10 仕様に準拠しています。JBoss EAP は高可用性クラスタリング、メッセージング、分散キャッシングなどの機能の事前設定オプションを提供します。必要時のみにサービスを有効にできるモジュラー構造が含まれるため、起動速度が改善されます。

Web ベースの管理コンソールと管理コマンドラインインターフェイス (CLI) を使用すると、タスクをスクリプト化して自動化し、XML 設定ファイルを編集する必要がなくなります。さらに、JBoss EAP には、セキュアでスケーラブルな Jakarta EE アプリケーションの開発、デプロイ、実行に使用できる API と開発フレームワークが組み込まれています。JBoss EAP 8.0 は、Web Profile、Core Profile、Full Platform 仕様の Jakarta EE 10 互換実装です。

1.1. OPENSHIFT での JBOSS EAP の仕組み

Red Hat は、OpenShift 上の JBoss EAP でアプリケーションイメージをビルドおよび実行するためのコンテナーイメージを提供します。



注記

Red Hat は、JBoss EAP を含むイメージを提供しなくなりました。

1.2. 比較: JBOSS EAP および JBOSS EAP FOR OPENSHIFT

JBoss EAP 製品と JBoss EAP for OpenShift イメージを比較すると、顕著な違いがいくつかあります。以下の表は、これらの違いを説明し、JBoss EAP for OpenShift の現在のバージョンに含まれる機能またはサポートされる機能を示します。

表1.1 JBoss EAP と JBoss EAP for OpenShift の違い

JBoss EAP の機能	JBoss EAP for OpenShift での状態	説明
JBoss EAP 管理コンソール	含まれない	本リリースの JBoss EAP for OpenShift には JBoss EAP 管理コンソールは含まれません。
JBoss EAP 管理 CLI	非推奨	JBoss EAP 管理 CLI は、コンテナー化環境で実行されている JBoss EAP との使用が推奨されません。管理 CLI を使用して実行中のコンテナーで変更した設定内容は、コンテナーの再起動時に失われます。トラブルシューティング目的の場合は、Pod 内から管理 CLI にアクセスできます。
マネージドドメイン	サポート対象外	JBoss EAP マネージドドメインはサポートされませんが、アプリケーションの作成および配布は OpenShift 上のコンテナーで管理されます。
デフォルトのルートページ	無効	デフォルトのルートページは無効になっていますが、独自のアプリケーションを **ROOT.war** としてルートコンテキストにデプロイできます。

JBoss EAP の機能	JBoss EAP for OpenShift での状態	説明
リモートメッセージング	サポート対象	inter-Pod およびリモートメッセージングの Red Hat AMQ はサポートされます。ActiveMQ Artemis は、JBoss EAP インスタンスとの単一 Pod 内のメッセージングに対してのみサポートされ、Red Hat AMQ が存在しない場合のみ有効になります。
トランザクションリカバリー	サポート対象	OpenShift 4 でテストおよびサポートされている唯一のトランザクションリカバリー方法は、EAP Operator です。EAP Operator を使用したトランザクションの回復の詳細は、JBoss EAP Operator による安全なトランザクションリカバリー を参照してください。

1.3. バージョンの互換性とサポート

JBoss EAP for OpenShift は、OpenJDK 17 のイメージを提供します。

イメージの 2 つのバリアントとして、S2I ビルダーイメージとランタイムイメージを使用できます。S2I Builder イメージには、S2I ビルド中に完全な JBoss EAP サーバーをプロビジョニングできるようにするために必要なすべてのツールが含まれています。ランタイムイメージには JBoss EAP の実行に必要な依存関係が含まれていますが、サーバーは含まれません。サーバーは、チェーンビルド時にランタイムイメージでインストールされます。

JBoss EAP 8.0 for OpenShift のイメージには、以下の変更が適用されています。

- S2I ビルダーイメージは、インストールされる JBoss EAP サーバーを含んでおらず、S2I ビルド中に JBoss EAP 8.0 サーバーをインストールします。
- S2I ビルド中に、アプリケーションの **pom** ファイルで eap-maven-plugin を設定します。
- S2I ビルド中に **GALLEON_PROVISION_FEATURE_PACKS**、**GALLEON_PROVISION_LAYERS**、および **GALLEON_PROVISION_CHANNELS** 環境変数を設定することで、既存の JBoss EAP 7.4 アプリケーションを変更せずに使用します。
- S2I ビルド中にプロビジョニングされた JBoss EAP サーバーには、OpenShift 用にカスタマイズされた **standalone.xml** サーバー設定ファイルが含まれています。

重要

サーバーには、JBoss EAP 7.4 で使用された **standalone-openshift.xml** 設定ファイルではなく、**standalone.xml** 設定ファイルが含まれています。

- イメージ内の **JBOSS_HOME** 値は **/opt/server** です。JBoss EAP 7.4 の場合、**JBOSS_HOME** の値は **/opt/eap** でした。
- **Jolokia エージェント** はイメージに表示されなくなりました。
- **Prometheus エージェント** がインストールされていません。

- **Python プローブ** はもう存在しません。
- **SSO** アダプターはイメージに存在しなくなりました。
- **activemq.rar** はもう存在しません。

注記

次の検出メカニズムプロトコルは廃止され、他のプロトコルに置き換えられました。

- **openshift.DNS_PING** プロトコルは非推奨となり、**dns.DNS_PING** プロトコルに置き換えられました。**customized standalone.xml** ファイルで **openshift.DNS_PING** プロトコルを参照している場合は、プロトコルを **dns.DNS_PING** プロトコルに置き換えてください。
- **openshift.KUBE_PING** 検索メカニズムプロトコルは非推奨となり、**kubernetes.KUBE_PING** プロトコルに置き換えられました。

1.3.1. OpenShift 4.x サポート

OpenShift 4.1 の変更は Jolokia へのアクセスに影響します。Open Java Console は OpenShift 4.x Web コンソールで利用できなくなりました。

以前のリリースの OpenShift では、プロキシー化された特定の kube-apiserver 要求が認証され、クラスターに渡されていました。この動作は安全ではないと見なされているため、この方法での Jolokia へのアクセスはサポート対象外になりました。

OpenShift コンソールのコードベースの変更により、Open Java Console へのリンクが利用できなくなりました。

1.3.2. IBM Z サポート

libartemis-native の s390x バリアントはイメージに含まれません。そのため、AIO に関連するいかなる設定も考慮されません。

- **journal-type**: **journal-type** を **ASYNCIO** に設定しても効果はありません。この属性の値は、起動時に **NIO** にデフォルト設定されます。
- **journal-max-io**: この属性は影響を受けません。
- **journal-store-enable-async-io**: この属性は影響を受けません。

1.3.2.1. OpenShift での JBoss EAP 7.4 から JBoss EAP 8.0 へのアップグレード

OpenShift において JBoss EAP 7.4 でインストールされたファイル **standalone.xml** は、JBoss EAP 8.0 以降と互換性がありません。OpenShift の JBoss EAP 8.0 以降のコンテナーを起動する前に、ファイルを変更して、名前を **standalone.xml** に変更する必要があります。

関連情報

- OpenShift で JBoss EAP 7.1 を JBoss EAP 8.0 にアップグレードする場合の **standalone.xml** の更新

1.3.3. デプロイ方法

EAP Operator を使用して、OpenShift に JBoss EAP Java アプリケーションをデプロイできます。EAP Operator は OpenShift API を拡張する JBoss EAP 専用のコントローラーであり、OpenShift ユーザーに代わって複雑なステートフルアプリケーションのインスタンスを作成、設定、管理します。

関連情報

- EAP Operator の詳細は、JBoss EAP Operator による OpenShift へのアプリケーションのデプロイの自動化 を参照してください。

第2章 JBOSS EAP 8.0 のパッケージ名前空間の変更

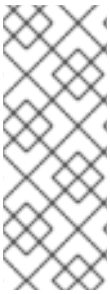
このセクションでは、JBoss EAP 8.0 のパッケージ名前空間の変更に関する追加情報を提供します。JBoss EAP 8.0 は、Jakarta EE 10 および Jakarta EE 10 API の他の多くの実装を完全にサポートします。JBoss EAP 8.0 の Jakarta EE 10 でサポートされる重要な変更点は、パッケージの名前空間の変更です。

2.1. JAVAX から JAKARTA への名前空間の変更

Jakarta EE 8 と EE 10 の主な違いは、EE API Java パッケージの名前が **javax.*** から **jakarta.*** に変更されたことです。これは、Java EE が Eclipse Foundation に移行し、Jakarta EE が確立されたことに続くものです。

アプリケーションを JBoss EAP 7 から JBoss EAP 8 に移行する際には、この名前空間変更への対応が最大のタスクとなります。アプリケーションを Java EE 10 に移行するには、次の手順を完了する必要があります。

- import ステートメントまたはその他のソースコードにおける EE API クラスの使用を **javax** パッケージから **jakarta** パッケージに更新します。
- **javax** で始まる EE 指定のシステムプロパティーまたはその他の設定プロパティーの名前を、**jakarta** で始まるものに更新します。
- **java.util.ServiceLoader** メカニズムを使用してブートストラップされる EE インターフェイスまたは抽象クラスのアプリケーション提供の実装がある場合は、実装クラスを識別するリソースの名前を **META-INF/services/javax.[rest_of_name]** から **META-INF/services/jakarta.[rest_of_name]** に変更します。



注記

Red Hat Migration Toolkit を使用すると、アプリケーションソースコード内の名前空間の更新が容易になります。詳細は、How to use Red Hat Migration Toolkit for Auto-Migration of an Application to the Jakarta EE 10 Namespace を参照してください。ソースコードを移行できない場合は、オープンソースの Eclipse Transformer プロジェクトで、既存の Java アーカイブを **javax** 名前空間から **jakarta** 名前空間に変換するバイトコード変換ツールが提供されています。

注記

この変更は、Java SE に含まれる **javax** パッケージには影響しません。

関連情報

- 詳細は、javax から jakarta パッケージ名前空間への変更 を参照してください。

第3章 OPENSHIFT CONTAINER PLATFORM での JBOSS EAP アプリケーションの構築と実行

source-to-image (S2I) プロセスに従って、JBoss EAP for OpenShift イメージで Java アプリケーションをビルドおよび実行できます。

3.1. 前提条件

- OpenShift インスタンスがインストールされ、操作可能になっています。

3.2. アプリケーションのデプロイに向けた OPENSHIFT の準備

JBoss EAP アプリケーション開発者は、アプリケーションを OpenShift にデプロイできます。次の例では、**kitchensink** クイックスタートが、Jakarta Server Faces、Jakarta Contexts and Dependency Injection、Jakarta Enterprise Beans、Jakarta Persistence、および Jakarta Bean Validation を使用して Jakarta EE の web 対応データベースアプリケーションを実行します。詳細は、JBoss EAP 8.0 の **kitchensink** クイックスタートを参照してください。以下の手順に従って、アプリケーションをデプロイします。

手順

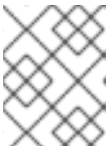
1. **oc login** コマンドを使用して、OpenShift インスタンスにログインします。
2. OpenShift でプロジェクトを作成します。
   次のコマンドを使用してプロジェクトを作成します。プロジェクトを使用すると、他のグループとは別にコンテンツを整理および管理できます。

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

   たとえば、以下のコマンドを使用して、**kitchensink** クイックスタートで **eap-demo** という名前のプロジェクトを作成します。

   ```
   $ oc new-project eap-demo
   ```

3. **任意の手順**: キーストアおよびシークレットを作成します。

   注記

   OpenShift プロジェクトで HTTPS 対応の機能を使用する場合は、キーストアとシークレットを作成する必要があります。

   a. 以下のように、Java **keytool** コマンドを使用して、キーストアを生成します。

警告

以下のコマンドは自己署名証明書を生成しますが、実稼働環境では信用性が確認された認証局 (CA) の独自の SSL 証明書を SSL で暗号化された接続 (HTTPS) に使用します。

```
$ keytool -genkey -keyalg RSA -alias <alias_name> -keystore
<keystore_filename.jks> -validity 360 -keysize 2048
```

たとえば、**kitchensink** クイックスタートでは、以下のコマンドを使用してキーストアを生成します。

```
$ keytool -genkey -keyalg RSA -alias eapdemo-selfsigned -keystore keystore.jks -validity
360 -keysize 2048
```

b. 次のコマンドを使用して、新しいキーストアからシークレットを作成します。

```
$ oc create secret generic <secret_name> --from-file=<keystore_filename.jks>
```

たとえば、**kitchensink** クイックスタートでは、以下のコマンドを使用してシークレットを作成します。

```
$ oc create secret generic eap-app-secret --from-file=keystore.jks
```

関連情報

- ImageStreams and Pods fail to pull images when Dev Portal generated secret is added in the namespace

3.3. OPENSHIFT で SOURCE-TO-IMAGE を使用したアプリケーションイメージのビルド

source-to-image (S2I) ワークフローに従って、JBoss EAP アプリケーションの再現可能なコンテナーイメージをビルドします。これらの生成されたコンテナーイメージには、アプリケーションのデプロイメントとすぐに実行できる JBoss EAP サーバーが含まれます。

S2I ワークフローは、Git リポジトリーからソースコードを取得し、使用する言語とフレームワークをベースとするコンテナーに挿入します。S2I ワークフローが完了すると、**src** コードがコンパイルされ、アプリケーションがパッケージ化されて JBoss EAP サーバーにデプロイされます。

詳細は、Legacy server provisioning for JBoss EAP S2I を参照してください。

注記

JBoss EAP では、Jakarta EE 10 を使用してアプリケーションを開発する場合にのみ S2I イメージを使用できます。

前提条件

- アクティブな Red Hat カスタマーアカウントを持っている。
- レジストリーサービスアカウントを持っている。Red Hat カスタマーポータルの手順に従い、レジストリーサービスアカウントを使用して認証トークンを作成 してください。
- Red Hat Ecosystem Catalog からイメージをプルするために使用できる OpenShift シークレット YAML ファイルをダウンロードしている。詳細は、OpenShift Secret を参照してください。
- **oc login** コマンドを使用して OpenShift にログインしている。
- Helm がインストールされている。詳細は、Installing Helm を参照してください。
- 管理 CLI で次のコマンドを入力して、JBoss EAP Helm チャートのリポジトリーをインストールしている。

```
$ helm repo add jboss-eap https://jbossas.github.io/eap-charts/
```

手順

1. 次の YAML コンテンツを使用して、**helm.yaml** という名前のファイルを作成します。

```
build:
  uri: https://github.com/jboss-developer/jboss-eap-quickstarts.git
  ref: EAP_8.0.0.GA
  contextDir: helloworld
deploy:
  replicas: 1
```

2. 以下のコマンドを使用して、JBoss EAP アプリケーションを OpenShift にデプロイします。

```
$ helm install helloworld -f helm.yaml jboss-eap/eap8
```

検証

- **curl** を使用してアプリケーションにアクセスします。

```
$ curl https://$(oc get route helloworld --template='{{ .spec.host }}')/HelloWorld
```

アプリケーションがデプロイされていれば、**Hello World!** が出力されます。

3.4. OPENSHIFT へのサードパーティーアプリケーションのデプロイ

コンパイル済みの WAR ファイルまたは EAR アーカイブを使用して、OpenShift デプロイ用のアプリケーションイメージを作成できます。Dockerfile を使用して、これらのアーカイブを、更新された包括的なランタイムスタック (オペレーティングシステム、Java、および JBoss EAP コンポーネントを含む) とともに JBoss EAP サーバーにデプロイします。

注記

Red Hat は、ビルド済みの JBoss EAP サーバーイメージを提供しません。

3.4.1. デフォルト設定での JBoss EAP サーバーのプロビジョニング

ビルダーイメージを使用すると、デフォルト設定で JBoss EAP サーバーを OpenShift にインストールして設定できます。シームレスなデプロイを実現するために、手順に従ってサーバーをプロビジョニングし、アプリケーションファイルを転送して、必要なカスタマイズを行ってください。

前提条件

- サポート対象の Red Hat JBoss Enterprise Application Platform コンテナーイメージにアクセスできる。以下に例を示します。
  - **registry.redhat.io/jboss-eap-8/eap8-openjdk17-builder-openshift-rhel8**
  - **registry.redhat.io/jboss-eap-8/eap8-openjdk17-runtime-openshift-rhel8**
- システムに podman がインストールされている。サポート対象の RHEL で利用可能な最新の podman バージョンを使用してください。詳細は、Red Hat JBoss Enterprise Application Platform 8.0 でサポートされる構成 を参照してください。

手順

1. 下記の Dockerfile の内容をコピーします。

```
# Use EAP 8 Builder image to create a JBoss EAP 8 server
# with its default configuration

FROM registry.redhat.io/jboss-eap-8/eap8-openjdk17-builder-openshift-rhel8:latest AS
builder

# Set up environment variables for provisioning. (1)
ENV GALLEON_PROVISION_FEATURE_PACKS org.jboss.eap:wildfly-ee-galleon-
pack,org.jboss.eap.cloud:eap-cloud-galleon-pack
ENV GALLEON_PROVISION_LAYERS cloud-default-config
# Specify the JBoss EAP version (2)
ENV GALLEON_PROVISION_CHANNELS org.jboss.eap.channels:eap-8.0


# Run the assemble script to provision the server.
RUN /usr/local/s2i/assemble

# Copy the JBoss EAP 8 server from the builder image to the runtime image.
FROM registry.redhat.io/jboss-eap-8/eap8-openjdk17-runtime-openshift-rhel8:latest AS
runtime

# Set appropriate ownership and permissions.
COPY --from=builder --chown=jboss:root $JBOSS_HOME $JBOSS_HOME

# Steps to add:
# (1) COPY the WAR/EAR to $JBOSS_HOME/standalone/deployments
#       with the jboss:root user. For example:
#     COPY --chown=jboss:root my-app.war $JBOSS_HOME/standalone/deployments (3)
# (2) (optional) server modification. You can modify EAP server configuration:
#
#       * invoke management operations. For example
#
#        RUN $JBOSS_HOME/bin/jboss-cli.sh --commands="embed-server,/system-
property=Foo:add(value=Bar)"
#
```

```
#       First operation must always be embed-server.
#
#      * copy a modified standalone.xml in $JBOSS_HOME/standalone/configuration/
#        for example
#
#     COPY --chown=jboss:root standalone.xml  $JBOSS_HOME/standalone/configuration

# Ensure appropriate permissions for the copied files.
RUN chmod -R ug+rwX $JBOSS_HOME
```

1. **MAVEN_MIRROR_URL** 環境変数を指定できます。この変数は、JBoss EAP Maven プラグインによってイメージ内で内部的に使用されます。詳細は、アーティファクトリポジトリーミラー を参照してください。
2. この Dockerfile は、マイナーリリース用に更新する必要はありません。特定のバージョンを使用する場合は、**GALLEON_PROVISION_CHANNELS** 環境変数で JBoss EAP バージョンを指定します。詳細は、環境変数 を参照してください。
3. コピーした Dockerfile を変更して、WAR ファイルをコンテナーに含めます。以下に例を示します。

```
COPY --chown=jboss:root <my-app.war> $JBOSS_HOME/standalone/deployments
```

<myapp.war> は、イメージに追加する Web アーカイブへのパスに置き換えます。

2. podman を使用してアプリケーションイメージをビルドします。

```
$ podman build -t my-app .
```

コマンドを実行すると、**my-app** コンテナーイメージを OpenShift にデプロイする準備が整います。

3. コンテナーイメージを次のいずれかにアップロードします。
   - OpenShift からアクセスできる内部レジストリー。
   - OpenShift レジストリー。イメージをビルドしたマシンからイメージを直接プッシュします。詳細は、How to push a container image into the image registry in RHOCP 4 を参照してください。
4. レジストリーからイメージをデプロイする場合は、Helm チャート、Operator、Deployment などのデプロイストラテジーを使用します。希望する方法を選択し、要件に応じてイメージの完全な URL か ImageStreams を使用します。詳細は、Helm チャートを使用した OpenShift 上での JBoss EAP アプリケーションのビルドおよびデプロイ を参照してください。

3.5. OPENID CONNECT を使用して OPENSHIFT 上の JBOSS EAP アプリケーションを保護する

JBoss EAP ネイティブ OpenID Connect (OIDC) クライアントを使用して、外部 OpenID プロバイダーを使用して認証を委任します。OIDC は、JBoss EAP などのクライアントが OpenID プロバイダーによって実行される認証に基づいてユーザーのアイデンティティーを検証できるようにするアイデンティティーレイヤーです。

elytron-oidc-client サブシステムと **elytron-oidc-client** Galleon レイヤーは、OpenID プロバイダーに

接続するために JBoss EAP でネイティブ OIDC クライアントを提供します。JBoss EAP は、OpenID プロバイダーの設定に基づいて、アプリケーションの仮想セキュリティードメインを自動的に作成します。

elytron-oidc-client サブシステムは、3 つの異なる方法で設定できます。

- **oidc.json** をデプロイメントに追加します。
- CLI スクリプトを実行して **elytron-oidc-client** サブシステムを設定します。
- OpenShift での JBoss EAP サーバーの起動時に **elytron-oidc-client** サブシステムを設定するための環境変数の定義。

注記

この手順では、環境変数を使用して **elytron-oidc-client** サブシステムを設定し、OIDC でアプリケーションを保護する方法について説明します。

3.5.1. JBoss EAP での OpenID Connect 設定

OpenID プロバイダーを使用してアプリケーションを保護する場合、セキュリティードメインリソースをローカルで設定する必要はありません。**elytron-oidc-client** サブシステムは、OpenID プロバイダーと接続するための JBoss EAP のネイティブ OpenID Connect (OIDC) クライアントを提供します。JBoss EAP は、OpenID プロバイダーの設定に基づいて、アプリケーションの仮想セキュリティードメインを自動的に作成します。

重要

OIDC クライアントは Red Hat build of Keycloak で使用してください。JSON Web トークン (JWT) であるアクセストークンを使用するように設定でき、RS256、RS384、RS512、ES256、ES384、または ES512 署名アルゴリズムを使用するように設定できる場合は、他の OpenID プロバイダーを使用できます。

OIDC の使用を有効にするには、**elytron-oidc-client** サブシステムまたはアプリケーション自体を設定できます。JBoss EAP は、次のように OIDC 認証をアクティブにします。

- アプリケーションを JBoss EAP にデプロイすると、**elytron-oidc-client** サブシステムがデプロイメントをスキャンして、OIDC 認証メカニズムが必要かどうかを検出します。
- サブシステムが **elytron-oidc-client** サブシステムまたはアプリケーションデプロイメント記述子のいずれかでデプロイメントの OIDC 設定を検出した場合、JBoss EAP はアプリケーションの OIDC 認証メカニズムを有効にします。
- サブシステムが両方の場所で OIDC 設定を検出した場合、**elytron-oidc-client** サブシステム **secure-deployment** 属性の設定が、アプリケーションデプロイメント記述子の設定よりも優先されます。

関連情報

- OpenID Connect specification
- OpenID Connect ライブラリー
- OpenID Connect と Red Hat build of Keycloak を使用したアプリケーションの保護

3.5.2. OpenID Connect で保護されたアプリケーションの作成

web-application を作成するには、必要な依存関係とディレクトリー構造で Maven プロジェクトを作成します。ログインユーザーのプリンシパルおよび属性から取得したユーザー名を返すサーブレットを含む Web アプリケーションを作成します。ログインしているユーザーがないと、サーブレットは「NO AUTHENTICATED USER」のテキストを返します。

前提条件

- Maven がインストールされている。詳細は、Downloading Apache Maven を参照してください。

手順

1. **mvn** コマンドを使用して Maven プロジェクトを設定します。このコマンドは、プロジェクトのディレクトリー構造と **pom.xml** 設定ファイルを作成します。

   構文

   ```
   $ mvn archetype:generate \
   -DgroupId=${group-to-which-your-application-belongs} \
   -DartifactId=${name-of-your-application} \
   -DarchetypeGroupId=org.apache.maven.archetypes \
   -DarchetypeArtifactId=maven-archetype-webapp \
   -DinteractiveMode=false
   ```

   例:

   ```
   $ mvn archetype:generate \
   -DgroupId=com.example.app \
   -DartifactId=simple-webapp-example \
   -DarchetypeGroupId=org.apache.maven.archetypes \
   -DarchetypeArtifactId=maven-archetype-webapp \
   -DinteractiveMode=false
   ```

2. 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。

   構文

   ```
   $ cd <name-of-your-application>
   ```

   例:

   ```
   $ cd simple-webapp-example
   ```

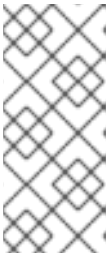
3. 生成された **pom.xml** ファイルの内容を、以下のテキストに置き換えます。

   ```
   <?xml version="1.0" encoding="UTF-8"?>

   <project xmlns="http://maven.apache.org/POM/4.0.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
   http://maven.apache.org/xsd/maven-4.0.0.xsd">
   ```

```
<modelVersion>4.0.0</modelVersion>

<groupId>com.example.app</groupId>
<artifactId>simple-webapp-example</artifactId>
<version>1.0-SNAPSHOT</version>
<packaging>war</packaging>

<name>simple-webapp-example Maven Webapp</name>
<!-- FIXME change it to the project's website -->
<url>http://www.example.com</url>

<properties>
  <maven.compiler.source>11</maven.compiler.source>
  <maven.compiler.target>11</maven.compiler.target>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  <version.maven.war.plugin>3.3.2</version.maven.war.plugin>
  <version.eap.plugin>1.0.0.Final-redhat-00014</version.eap.plugin>
  <version.server>8.0.0.GA-redhat-00009</version.server>
  <version.bom.ee>${version.server}</version.bom.ee>
</properties>


<repositories>
  <repository>
      <id>jboss</id>
      <url>https://maven.repository.redhat.com/ga/</url>
      <snapshots>
          <enabled>false</enabled>
      </snapshots>
  </repository>
</repositories>

<pluginRepositories>
  <pluginRepository>
      <id>jboss</id>
      <url>https://maven.repository.redhat.com/ga/</url>
      <snapshots>
          <enabled>false</enabled>
      </snapshots>
  </pluginRepository>
</pluginRepositories>

<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.jboss.bom</groupId>
      <artifactId>jboss-eap-ee-with-tools</artifactId>
      <version>${version.bom.ee}</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>

<dependencies>
  <dependency>
```

```
      <groupId>jakarta.servlet</groupId>
      <artifactId>jakarta.servlet-api</artifactId>
      <scope>provided</scope>
    </dependency>
    <dependency>
      <groupId>org.wildfly.security</groupId>
      <artifactId>wildfly-elytron-auth-server</artifactId>
    </dependency>
  </dependencies>

<build>
    <finalName>${project.artifactId}</finalName>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-war-plugin</artifactId>
        <version>${version.maven.war.plugin}</version>
      </plugin>
      <plugin>
        <groupId>org.jboss.eap.plugins</groupId>
        <artifactId>eap-maven-plugin</artifactId>
        <version>${version.eap.plugin}</version>
        <configuration>
          <channels>
            <channel>
              <manifest>
                <groupId>org.jboss.eap.channels</groupId>
                <artifactId>eap-8.0</artifactId>
              </manifest>
            </channel>
          </channels>
          <feature-packs>
            <feature-pack>
              <location>org.jboss.eap:wildfly-ee-galleon-pack</location>
            </feature-pack>
            <feature-pack>
              <location>org.jboss.eap.cloud:eap-cloud-galleon-pack</location>
            </feature-pack>
          </feature-packs>
          <layers>
            <layer>cloud-server</layer>
            <layer>elytron-oidc-client</layer>
          </layers>
          <galleon-options>
              <jboss-fork-embedded>true</jboss-fork-embedded>
          </galleon-options>
        </configuration>
        <executions>
          <execution>
            <goals>
              <goal>package</goal>
            </goals>
          </execution>
        </executions>
      </plugin>
```

```
    </plugins>
</build>
</project>
```



注記

- **<version.eap.plugin>1.0.0.Final-redhat-00014</version.eap.plugin>** は、JBoss EAP Maven プラグインのバージョンの例です。JBoss EAP Maven プラグインのリリースの詳細は、Red Hat Maven リポジトリー (https://maven.repository.redhat.com/earlyaccess/all/org/jboss/eap/plugins/e maven-plugin/) を参照してください。

4. Java ファイルを保存するディレクトリーを作成します。

   構文

   ```
   $ mkdir -p src/main/java/<path_based_on_artifactID>
   ```

   例:

   ```
   $ mkdir -p src/main/java/com/example/app
   ```

5. 新しいディレクトリーに移動します。

   構文

   ```
   $ cd src/main/java/<path_based_on_artifactID>
   ```

   例:

   ```
   $ cd src/main/java/com/example/app
   ```

6. 以下の内容で **SecuredServlet.java** ファイルを作成します。

```
package com.example.app;

import java.io.IOException;
import java.io.PrintWriter;
import java.security.Principal;
import java.util.ArrayList;
import java.util.Collection;
import java.util.Iterator;
import java.util.List;
import java.util.Set;

import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import org.wildfly.security.auth.server.SecurityDomain;
import org.wildfly.security.auth.server.SecurityIdentity;
```

```
import org.wildfly.security.authz.Attributes;
import org.wildfly.security.authz.Attributes.Entry;
/**
 * A simple secured HTTP servlet. It returns the user name and
 * attributes obtained from the logged-in user's Principal. If
 * there is no logged-in user, it returns the text
 * "NO AUTHENTICATED USER".
 */

@WebServlet("/secured")
public class SecuredServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        try (PrintWriter writer = resp.getWriter()) {

         Principal user = req.getUserPrincipal();
         SecurityIdentity identity = SecurityDomain.getCurrent().getCurrentSecurityIdentity();
         Attributes identityAttributes = identity.getAttributes();
         Set <String> keys = identityAttributes.keySet();
         String attributes = "<ul>";

         for (String attr : keys) {
          attributes += "<li> " +  attr + " : " + identityAttributes.get(attr).toString() + "</li>";
         }

         attributes+="</ul>";
         writer.println("<html>");
         writer.println("  <head><title>Secured Servlet</title></head>");
         writer.println("  <body>");
         writer.println("    <h1>Secured Servlet</h1>");
         writer.println("    <p>");
         writer.print(" Current Principal '");
         writer.print(user != null ? user.getName() : "NO AUTHENTICATED USER");
         writer.print("'");
         writer.print(user != null ? "\n" + attributes : "");
         writer.println("    </p>");
         writer.println("  </body>");
         writer.println("</html>");
        }
    }

}
```

7. アプリケーションの **web.xml** を設定して、アプリケーションリソースを保護します。

例:

```
<?xml version="1.0" encoding="UTF-8"?>

<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
    metadata-complete="false">
```

```
    <security-constraint>
        <web-resource-collection>
            <web-resource-name>secured</web-resource-name>
            <url-pattern>/secured</url-pattern>
        </web-resource-collection>

        <auth-constraint>
            <role-name>Users</role-name>
        </auth-constraint>
    </security-constraint>

    <login-config>
        <auth-method>OIDC</auth-method>
    </login-config>

    <security-role>
        <role-name>*</role-name>
    </security-role>
</web-app>
```

この例では、**Users** ロールを持つユーザーのみがアプリケーションにアクセスできます。

3.5.3. アプリケーションを OpenShift にデプロイする

JBoss EAP アプリケーション開発者は、OpenID Connect サブシステムを使用する OpenShift にアプリケーションをデプロイし、そのアプリケーションを Red Hat build of Keycloak サーバーと統合できます。以下の手順に従って、アプリケーションをデプロイします。

前提条件

以下の設定で、OpenShift で Red Hat build of Keycloak サーバーを設定している。詳細は、Red Hat build of Keycloak Operator を参照してください。

- **JBossEAP** というレルムを作成します。
- **demo** というユーザーを作成します。
- **demo** というユーザーのパスワードを設定します。**Temporary** を **OFF** に切り替えて、**Set Password** をクリックします。確認プロンプトで、**Set password** をクリックします。
- **Users** というロールを作成します。
- ロール **Users** をユーザー **demo** に割り当てます。
- **Client Roles** フィールドで、JBoss EAP 用に設定した **realm-management** を選択します。
- ロール **create-client** をクライアント **realm-management** に割り当てます。

手順

1. アプリケーションコードを Git リポジトリーにデプロイします。
2. OIDC 設定を含むシークレットを作成します。
    a. 次の内容を使用して、**oidc-secret.yaml** という名前のファイルを作成します。

```
apiVersion: v1
kind: Secret
metadata:
  name: oidc-secret
type: Opaque
stringData:
  OIDC_PROVIDER_NAME: rh-sso
  OIDC_USER_NAME: demo
  OIDC_USER_PASSWORD: demo
  OIDC_SECURE_DEPLOYMENT_SECRET: mysecret
```

b. 以下のコマンドを実行してシークレットを作成します。

```
$ oc apply -f oidc-secret.yaml
```

3. 次の内容を使用して、**helm.yaml** という名前のファイルを作成します。

```
build:
  uri: [URL TO YOUR GIT REPOSITORY]
deploy:
  envFrom:
 - secretRef:
      name: oidc-secret
```

4. JBoss EAP Helm チャートを使用してサンプルアプリケーションをデプロイします。

```
$ helm install eap-oidc-test-app -f helm.yaml jboss-eap/eap8
```

5. 環境変数を **oidc-secret.yaml** ファイルに追加して、OIDC プロバイダーの URL とアプリケーションのホスト名を設定します。

```
yaml
stringData:
  ...
  OIDC_HOSTNAME_HTTPS: <host of the application>
  OIDC_PROVIDER_URL: https://<host of the SSO provider>/realms/JBossEAP
```

OIDC_HOSTNAME_HTTPS の値は、次の出力に対応します。

```
echo $(oc get route eap-oidc-test-app --template='{{ .spec.host }}')
```

OIDC_PROVIDER_URL の値は、次の出力に対応します。

```
echo https://$(oc get route sso --template='{{ .spec.host }}')/realms/JBossEAP
```

OIDC_HOSTNAME_HTTP(S) が設定されていない場合は、ルート検出が試行されます。ルート検出を有効にするには、OpenShift ユーザーが **route** リソースを一覧表示できる必要があります。たとえば、**routeview** ロールを作成して **view** ユーザーに関連付けるには、次の **oc** コマンドを使用します。

```
$ oc create role <role-name> --verb=list --resource=route

$ oc adm policy add-role-to-user <role-name> <user-name> --role-namespace=<your
```

```
namespace>
```

6. **oc apply -f oidc-secret.yaml** でシークレットを更新します。
7. アプリケーションを再度デプロイして、OpenShift が新しい環境変数を使用するようにします。

```
$ oc rollout restart deploy eap-oidc-test-app
```

検証

1. ブラウザーで **https://<eap-oidc-test-app route>/** に移動します。
Red Hat build of Keycloak のログインページにリダイレクトされます。
2. 保護されたサーブレットにアクセスします。
3. 次の認証情報でログインします。

```
username: demo
password: demo
```

Principal ID を含むページが表示されます。

3.5.4. 環境変数ベースの設定

これらの環境変数を使用して、OpenShift イメージで JBoss EAP OIDC サポートを設定します。

表3.1 環境変数

環境変数	従来の SSO 環境変数	説明	必須	デフォルト値
OIDC_PROVIDER_NAME	なし。**SSO_*** 環境変数を使用すると、"rh-sso" という名前が内部的に設定されます。	OIDC_PROVIDER_NAME 変数を使用する場合は、**rh-sso** に設定する必要があります。	はい	
OIDC_PROVIDER_URL	**$SSO_URL/realms/$SSO_REALM**	プロバイダーの URL。	はい	
OIDC_USER_NAME	SSO_USERNAME	動的クライアント登録では、トークンを受け取るためにユーザー名が必要です。	はい	
OIDC_USER_PASSWORD	SSO_PASSWORD	動的クライアント登録では、トークンを受け取るためにユーザーパスワードが必要です。	はい	

環境変数	従来の SSO 環境変数	説明	必須	デフォルト値
OIDC_SECURE_DEPLOYMENT_SECRET	SSO_SECRET	これは、secure-deployment サブシステムと認証サーバークライアントの両方に認識されます。	いいえ	
OIDC_SECURE_DEPLOYMENT_PRINCIPAL_ATTRIBUTE	SSO_PRINCIPAL_ATTRIBUTE	プリンシパル名の値を設定します。	いいえ	**rh-sso** のデフォルトは **sub** (ID トークン) です。 典型的な値: preferred_username。
OIDC_SECURE_DEPLOYMENT_ENABLE_CORS	SSO_ENABLE_CORS	Single Sign-On アプリケーションの CORS を有効にします。	いいえ	デフォルトは **False** です。
OIDC_SECURE_DEPLOYMENT_BEARER_ONLY	SSO_BEARER_ONLY	ベアラートークンのみを受け入れ、ログ記録をサポートしないデプロイ。	いいえ	デフォルトは **False** です。
OIDC_PROVIDER_SSL_REQUIRED	NONE	デフォルトはプライベートアドレスやローカルアドレスなどの外部ですが、https はサポートしていません。	いいえ	外部
OIDC_PROVIDER_TRUSTSTORE	SSO_TRUSTSTORE	レルム **trustore** ファイルを指定します。設定されていない場合、アダプターは HTTPS 要求の処理時にトラストマネージャーを使用できません。	いいえ	
OIDC_PROVIDER_TRUSTSTORE_DIR	SSO_TRUSTSTORE_DIR	レルム **truststore** を検索するディレクトリー。設定されていない場合、アダプターは HTTPS 要求の処理時にトラストマネージャーを使用できません。	いいえ	

環境変数	従来の SSO 環境変数	説明	必須	デフォルト値
OIDC_PROVIDER_TRUSTSTORE_PASSWORD	SSO_TRUSTSTORE_PASSWORD	レルム **truststore** のパスワードを指定します。設定されていない場合、アダプターは HTTPS 要求の処理時にトラストマネージャーを使用できません。	いいえ	
OIDC_PROVIDER_TRUSTSTORE_CERTIFICATE_ALIAS	SSO_TRUSTSTORE_CERTIFICATE_ALIAS	レルム **trustore** エイリアスを指定します。クライアントを登録するには、認証サーバーと対話する必要があります。	いいえ	
OIDC_DISABLE_SSL_CERTIFICATE_VALIDATION	SSO_DISABLE_SSL_CERTIFICATE_VALIDATION	認証サーバーとやり取りしてクライアントを登録するときは、証明書の検証を無効にします。	いいえ	
OIDC_HOSTNAME_HTTP	HOSTNAME_HTTP	安全でないルートに使用されるホスト名。	いいえ	ルートが検出されます。
OIDC_HOSTNAME_HTTPS	HOSTNAME_HTTPS	保護されたルートに使用されるホスト名。	いいえ	保護されたルートが検出されます。
NONE	SSO_PUBLIC_KEY	シングルサインオンレルムの公開鍵。このオプションは使用されません。公開鍵は OIDC サブシステムによって自動的に取得されます。	いいえ	設定すると、このオプションが無視されているという警告が表示されます。

3.6. SAML を使用したアプリケーションの保護

Security Assertion Markup Language (SAML) は、2 者間での認証および認可情報の交換を可能にするデータ形式およびプロトコルです。通常、この 2 者には、アイデンティティプロバイダーとサービスプロバイダーが含まれます。この情報は、アサーションを含む SAML トークンの形式をとります。アイデンティティープロバイダーは、この SAML トークンをサブジェクトに発行して、サブジェクトがサービスプロバイダーで認証できるようにします。サブジェクトは複数のサービスプロバイダーで SAML トークンを再利用できるため、SAML v2 によるブラウザーベースのシングルサインオンが可能になります。

Keycloak SAML アダプター機能パックが提供する Galleon レイヤーを使用すると、Web アプリケーションを保護できます。

Keycloak SAML アダプター機能パックについては、SAML を使用してアプリケーションを保護するための Keycloak SAML アダプター機能パック を参照してください。

3.6.1. SAML を使用してアプリケーションを保護するための Keycloak SAML アダプター機能パック

Keycloak SAML アダプター Galleon パックは、**keycloak-saml** レイヤーを含む Galleon 機能パックです。この機能パックの **keycloak-saml** レイヤーを使用して、必要なモジュールと設定を JBoss EAP にインストールします。これらのモジュールと設定は、SAML の使用時にシングルサインオン (SSO) のアイデンティティプロバイダーとして Red Hat build of Keycloak を使用する場合に必要です。source-to-image (S2I) に **keycloak-saml** SAML アダプター Galleon レイヤーを使用する場合は、必要に応じて、Red Hat build of Keycloak などのアイデンティティーサービスプロバイダー (IDP) への自動登録を可能にする SAML クライアント機能を使用できます。

3.6.2. Red Hat build of Keycloak を OpenShift の SAML プロバイダーとして設定する

Red Hat build of Keycloak は、シングルサインオン (SSO) を使用して Web アプリケーションを保護するためのアイデンティティーおよびアクセス管理プロバイダーです。OAuth 2.0 の拡張機能である OpenID Connect と、SAML をサポートします。

次の手順は、SAML を使用してアプリケーションを保護するために必要な重要な手順を示しています。詳細は、Red Hat build of Keycloak のドキュメント を参照してください。

前提条件

- Red Hat build of Keycloak への管理者アクセス権を持っている。
- Red Hat build of Keycloak が実行中である。詳細は、Red Hat build of Keycloak Operator を参照してください。
- **oc login** コマンドを使用して OpenShift にログインしている。

手順

1. シングルサインオンレルム、ユーザー、およびロールを作成します。
2. Java **keytool** コマンドを使用してキーと証明書を生成します。

```
keytool -genkeypair -alias saml-app -storetype PKCS12 -keyalg RSA -keysize 2048 -
keystore keystore.p12 -storepass password -dname "CN=saml-basic-auth,OU=EAP SAML
Client,O=Red Hat EAP QE,L=MB,S=Milan,C=IT" -ext ku:c=dig,keyEncipherment -validity 365
```

3. キーストアを Java KeyStore (JKS) 形式でインポートします。

```
keytool -importkeystore -deststorepass password -destkeystore keystore.jks -srckeystore
keystore.p12 -srcstoretype PKCS12 -srcstorepass password
```

4. OpenShift でキーストアのシークレットを作成します。

```
$ oc create secret generic saml-app-secret --from-file=keystore.jks=./keystore.jks --
type=opaque
```

注記

この手順は、自動 SAML クライアント登録機能を使用する場合にのみ必要です。JBoss EAP が新しい SAML クライアントを **client-admin** ユーザーとして Red Hat build of Keycloak に登録する場合、JBoss EAP は新しい SAML クライアントの証明書を Red Hat build of Keycloak のクライアント設定に保存する必要があります。これにより、JBoss EAP は秘密鍵を保持しながら、公開証明書のみを Red Hat build of Keycloak に保存できるようになり、Red Hat build of Keycloak との通信用の認証済みクライアントが確立されます。

3.6.3. SAML で保護されたアプリケーションの作成

Security Assertion Markup Language (SAML) を使用すると、Web アプリケーションのセキュリティーを強化できます。SAML は、シングルサインオン (SSO) 機能とともに効果的なユーザー認証および認可を提供するため、Web アプリケーションを強化するときには頼りになる選択肢となります。

前提条件

- Maven がインストールされている。詳細は、Downloading Apache Maven を参照してください。

手順

1. **mvn** コマンドを使用して Maven プロジェクトをセットアップします。このコマンドで、プロジェクトのディレクトリー構造と **pom.xml** 設定ファイルの両方を作成します。

   構文

   ```
   $ mvn archetype:generate \
   -DgroupId=${group-to-which-your-application-belongs} \
   -DartifactId=${name-of-your-application} \
   -DarchetypeGroupId=org.apache.maven.archetypes \
   -DarchetypeArtifactId=maven-archetype-webapp \
   -DinteractiveMode=false
   ```

   例:

   ```
   $ mvn archetype:generate \
   -DgroupId=com.example.app \
   -DartifactId=simple-webapp-example \
   -DarchetypeGroupId=org.apache.maven.archetypes \
   -DarchetypeArtifactId=maven-archetype-webapp \
   -DinteractiveMode=false
   ```

2. 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。

   構文

   ```
   $ cd <name-of-your-application>
   ```

   例:

   ```
   $ cd simple-webapp-example
   ```

3. 生成された **pom.xml** ファイルの内容を、以下のテキストに置き換えます。

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.example.app</groupId>
  <artifactId>simple-webapp-example</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>war</packaging>

  <name>simple-webapp-example Maven Webapp</name>
  <!-- FIXME change it to the project's website -->
  <url>http://www.example.com</url>

  <properties>
    <maven.compiler.source>11</maven.compiler.source>
    <maven.compiler.target>11</maven.compiler.target>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <version.maven.war.plugin>3.3.2</version.maven.war.plugin>
    <version.eap.plugin>1.0.0.Final-redhat-00014</version.eap.plugin>
    <version.server>8.0.0.GA-redhat-00009</version.server>
    <version.bom.ee>${version.server}</version.bom.ee>
  </properties>

  <repositories>
    <repository>
        <id>jboss</id>
        <url>https://maven.repository.redhat.com/ga/</url>
        <snapshots>
            <enabled>false</enabled>
        </snapshots>
    </repository>
  </repositories>

  <pluginRepositories>
    <pluginRepository>
        <id>jboss</id>
        <url>https://maven.repository.redhat.com/ga/</url>
        <snapshots>
            <enabled>false</enabled>
        </snapshots>
    </pluginRepository>
  </pluginRepositories>

  <dependencyManagement>
    <dependencies>
      <dependency>
        <groupId>org.jboss.bom</groupId>
        <artifactId>jboss-eap-ee-with-tools</artifactId>
        <version>${version.bom.ee}</version>
        <type>pom</type>
        <scope>import</scope>
```

```
    </dependency>
  </dependencies>
</dependencyManagement>

<dependencies>
  <dependency>
    <groupId>jakarta.servlet</groupId>
    <artifactId>jakarta.servlet-api</artifactId>
    <scope>provided</scope>
  </dependency>
  <dependency>
    <groupId>org.wildfly.security</groupId>
    <artifactId>wildfly-elytron-auth-server</artifactId>
  </dependency>
</dependencies>

<build>
  <finalName>${project.artifactId}</finalName>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-war-plugin</artifactId>
      <version>${version.maven.war.plugin}</version>
    </plugin>
    <plugin>
      <groupId>org.jboss.eap.plugins</groupId>
      <artifactId>eap-maven-plugin</artifactId>
      <version>${version.eap.plugin}</version>
      <configuration>
        <channels>
          <channel>
            <manifest>
              <groupId>org.jboss.eap.channels</groupId>
              <artifactId>eap-8.0</artifactId>
            </manifest>
          </channel>
        </channels>
        <feature-packs>
          <feature-pack>
            <location>org.jboss.eap:wildfly-ee-galleon-pack</location>
          </feature-pack>
          <feature-pack>
            <location>org.jboss.eap.cloud:eap-cloud-galleon-pack</location>
          </feature-pack>
          <feature-pack>
            <location>org.keycloak:keycloak-saml-adapter-galleon-pack</location>
          </feature-pack>
        </feature-packs>
        <layers>
          <layer>cloud-server</layer>
          <layer>keycloak-saml</layer>
        </layers>
        <galleon-options>
          <jboss-fork-embedded>true</jboss-fork-embedded>
        </galleon-options>
      </configuration>
```

```
            <executions>
              <execution>
                <goals>
                  <goal>package</goal>
                </goals>
              </execution>
            </executions>
          </plugin>
        </plugins>
      </build>
    </project>
```

注記

- **<version.eap.plugin>1.0.0.Final-redhat-00014</version.eap.plugin>** は、JBoss EAP Maven プラグインのバージョンの例です。JBoss EAP Maven プラグインのリリースの詳細は、Red Hat Maven リポジトリー (https://maven.repository.redhat.com/earlyaccess/all/org/jboss/eap/plugins/e maven-plugin/) を参照してください。

4. Java ファイルを保存するディレクトリーを作成します。

   構文

   ```
   $ mkdir -p src/main/java/<path_based_on_artifactID>
   ```

   例:

   ```
   $ mkdir -p src/main/java/com/example/app
   ```

5. 新しいディレクトリーに移動します。

   構文

   ```
   $ cd src/main/java/<path_based_on_artifactID>
   ```

   例:

   ```
   $ cd src/main/java/com/example/app
   ```

6. 次の設定を含む **SecuredServlet.java** という名前のファイルを作成します。

   ```
   package com.example.app;

   import java.io.IOException;
   import java.io.PrintWriter;
   import java.security.Principal;
   import java.util.Set;

   import jakarta.servlet.ServletException;
   import jakarta.servlet.annotation.WebServlet;
   import jakarta.servlet.http.HttpServlet;
   ```

```
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import org.wildfly.security.auth.server.SecurityDomain;
import org.wildfly.security.auth.server.SecurityIdentity;
import org.wildfly.security.authz.Attributes;
/**
 * A simple secured HTTP servlet. It returns the user name and
 * attributes obtained from the logged-in user's Principal. If
 * there is no logged-in user, it returns the text
 * "NO AUTHENTICATED USER".
 */

@WebServlet("/secured")
public class SecuredServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        try (PrintWriter writer = resp.getWriter()) {

         Principal user = req.getUserPrincipal();
         SecurityIdentity identity = SecurityDomain.getCurrent().getCurrentSecurityIdentity();
         Attributes identityAttributes = identity.getAttributes();
         Set <String> keys = identityAttributes.keySet();
         String attributes = "<ul>";

         for (String attr : keys) {
          attributes += "<li> " +  attr + " : " + identityAttributes.get(attr).toString() + "</li>";
         }

         attributes+="</ul>";
         writer.println("<html>");
         writer.println("  <head><title>Secured Servlet</title></head>");
         writer.println("  <body>");
         writer.println("    <h1>Secured Servlet</h1>");
         writer.println("    <p>");
         writer.print(" Current Principal '");
         writer.print(user != null ? user.getName() : "NO AUTHENTICATED USER");
         writer.print("'");
         writer.print(user != null ? "\n" + attributes : "");
         writer.println("    </p>");
         writer.println("  </body>");
         writer.println("</html>");
        }
    }

}
```

7. **web.xml** ファイルのディレクトリー構造を作成します。

```
mkdir -p src/main/webapp/WEB-INF
cd src/main/webapp/WEB-INF
```

8. アプリケーションの **web.xml** ファイルを設定して、アプリケーションのリソースを保護します。

例:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
    metadata-complete="false">

    <security-constraint>
        <web-resource-collection>
            <web-resource-name>secured</web-resource-name>
            <url-pattern>/secured</url-pattern>
        </web-resource-collection>

        <auth-constraint>
            <role-name>user</role-name>
        </auth-constraint>
    </security-constraint>

    <login-config>
        <auth-method>KEYCLOAK-SAML</auth-method>
    </login-config>

    <security-role>
        <role-name>user</role-name>
    </security-role>
</web-app>
```

この例では、**user** ロールを持つユーザーのみがアプリケーションにアクセスできます。

検証

アプリケーションを作成したら、リモート Git リポジトリーにコミットします。

1. Git リポジトリーを作成します (**https://github.com/your-username/simple-webapp-example** など)。リモートリポジトリーと Git の詳細は、Getting started with Git - About remote repositories を参照してください。
2. アプリケーションのルートフォルダーから、次の Git コマンドを実行します。

```
git init -b main
git add pom.xml src
git commit -m "First commit"
git remote add origin git@github.com:your-username/simple-webapp-example.git
git remote -v
git push -u origin main
```

この手順により、アプリケーションがリモートリポジトリーにコミットされ、オンラインでアクセスできるようになります。

3.6.4. OpenShift 上での SAML で保護されたアプリケーションのビルドとデプロイ

JBoss EAP および Single Sign-On (SSO) Galleon レイヤーを使用して、SAML で保護されたアプリケーションを OpenShift 上でビルドおよびデプロイできます。

前提条件

- Helm がインストールされている。詳細は、Installing Helm を参照してください。
- SAML アプリケーションプロジェクトを作成し、Git リポジトリーでアクセスできるようにしている。
- 管理 CLI で次のコマンドを入力して、JBoss EAP Helm チャートのリポジトリーをインストールしている。

```
$ helm repo add jboss-eap https://jbossas.github.io/eap-charts/
```

手順

1. アプリケーションコードを Git リポジトリーにデプロイします。
2. 必要な環境変数を含む OpenShift シークレットを作成します。

```
apiVersion: v1
kind: Secret
metadata:
  name: saml-secret
type: Opaque
stringData:
  SSO_REALM: "saml-basic-auth"
  SSO_USERNAME: "client-admin"
  SSO_PASSWORD: "client-admin"
  SSO_SAML_CERTIFICATE_NAME: "saml-app"
  SSO_SAML_KEYSTORE: "keystore.jks"
  SSO_SAML_KEYSTORE_PASSWORD: "password"
  SSO_SAML_KEYSTORE_DIR: "/etc/sso-saml-secret-volume"
  SSO_SAML_LOGOUT_PAGE: "/simple-webapp-example"
  SSO_DISABLE_SSL_CERTIFICATE_VALIDATION: "true"
```

3. 指定の YAML コンテンツを **saml-secret.yaml** などのファイルに保存します。
4. 次のコマンドを使用して、保存した YAML ファイルを適用します。

```
oc apply -f saml-secret.yaml
```

5. 次の設定を含む **helm.yaml** という名前のファイルを作成します。

```
build:
  uri: [WEB ADDRESS TO YOUR GIT REPOSITORY]
deploy:
  volumes:
    - name: saml-keystore-volume
      secret:
        secretName: saml-app-secret
  volumeMounts:
    - name: saml-keystore-volume
      mountPath: /etc/sso-saml-secret-volume
      readOnly: true
```

```
envFrom:
  - secretRef:
      name: saml-secret
```

注記

Web アドレスは HTTP 形式で指定してください (**http://www.redhat.com** など)。Maven ミラーを使用している場合は、次のように Web アドレスを指定します。

```
build:
  uri: [WEB ADDRESS TO YOUR GIT REPOSITORY]
  env:
    - name: "MAVEN_MIRROR_URL"
      value: "http://..."
```

6. JBoss EAP Helm チャートを使用してサンプルアプリケーションをデプロイします。

```
$ helm install saml-app -f helm.yaml jboss-eap/eap8
```

7. 環境変数を **saml-secret.yaml** ファイルに追加して、Keycloak サーバー URL とアプリケーションルートを設定します。

```
stringData:
  ...
  HOSTNAME_HTTPS: <saml-app application route>
  SSO_URL: https://<host of the Keycloak server>
```

<saml-app application root> と **<host of the Keycloak server>** は、適切な値に置き換えます。

HOSTNAME_HTTPS の値は、次の出力に対応します。

```
echo $(oc get route saml-app --template='{{ .spec.host }}')
```

SSO_URL の値は、次の出力に置き換えます。

```
echo https://$(oc get route sso --template='{{ .spec.host }}')
```

注記

このコマンドを使用できない場合は、**oc get routes** を使用して利用可能なルートをリストし、Red Hat build of Keycloak インスタンスへのルートを選択してください。

8. **oc apply -f saml-secret.yaml** を使用してシークレットを更新します。

検証

1. OpenShift で新しい環境変数が使用されるように、アプリケーションを再度デプロイします。

```
$ oc rollout restart deploy saml-app
```

2. ブラウザーで、アプリケーションの URL に移動します。たとえば、**https://<saml-app route>/simple-webapp-example** です。
Red Hat build of Keycloak のログインページにリダイレクトされます。

3. Web アドレスを取得するために、次のコマンドを使用して保護されたサーブレットにアクセスします。

```
echo https://$(oc get route saml-app --template='{{ .spec.host }}')/simple-webapp-example/secured
```

4. 次の認証情報でログインします。

```
username: demo
password: demo
```

プリンシパル ID を含むページが表示されます。

これで、アプリケーションが SAML を使用して保護されました。

3.6.5. SSO レルム、ユーザー、およびロールの作成

Red Hat build of Keycloak 環境では、シングルサインオン (SSO) レルムの設定、ユーザーロールの定義、アクセス制御の管理を実行できます。これらの操作により、セキュリティーの強化、ユーザーアクセス管理の簡略化、認証エクスペリエンスの効率化を実現できます。これは、SSO 設定を最適化し、ユーザー認証プロセスを改善するために不可欠です。

前提条件

- Red Hat build of Keycloak への管理者アクセス権を持っている。
- Red Hat build of Keycloak が実行中である。

手順

1. URL **https://<SSO route>/** を使用して Red Hat build of Keycloak 管理コンソールにログインします。
2. Red Hat build of Keycloak でレルムを作成します (**saml-basic-auth** など)。その後、このレルムを使用して、必要なユーザー、ロール、およびクライアントを作成できます。
詳細は、レルムの作成 を参照してください。
3. **saml-basic-auth** レルム内にロールを作成します。たとえば、**user** などです。
詳細は、レルムロールの作成 を参照してください。
4. ユーザーを作成します。たとえば、**demo** などです。
詳細は、ユーザーの作成 を参照してください。
5. ユーザーのパスワードを作成します。たとえば、**demo** などです。
パスワードが一時的なものでないことを確認してください。詳細は、ユーザーのパスワードの設定 を参照してください。
6. ログインアクセス用の **user** ロールを **demo** ユーザーに割り当てます。
詳細は、ロールマッピングの割り当て を参照してください。

7. ユーザーを作成します。たとえば、**client-admin** などです。
JBoss EAP サーバーの起動時に Keycloak サーバーに SAML クライアントを作成するには、**client-admin** ユーザーを使用できます。このユーザーには追加の権限が必要です。詳細は、ユーザーの作成 を参照してください。

8. ユーザーのパスワードを作成します。たとえば、**client-admin** などです。
パスワードが一時的なものでないことを確認してください。詳細は、ユーザーのパスワードの設定 を参照してください。

9. **Client Roles** ドロップダウンリストから **realm-management** を選択します。

10. **create-client**、**manage-clients**、および **manage-realm** ロールを **client-admin** ユーザーに割り当てます。
詳細は、ロールマッピングの割り当て を参照してください。

3.6.6. SAML サブシステムを設定するための環境変数

次の変数を理解して使用することで、環境内での Keycloak サーバーの統合を最適化できます。これにより、アプリケーション用にシームレスでセキュアな Keycloak セットアップを実現できます。

表3.2 環境変数

環境変数	説明	必須
APPLICATION_NAME	デプロイメント名から導出されるクライアント名の接頭辞として使用されます。	オプション
HOSTNAME_HTTP	HTTP OpenShift ルートのカスタムの **hostname**。設定されていない場合は、ルート検出が実行されます。	オプション
HOSTNAME_HTTPS	HTTPS OpenShift ルートのカスタムの **hostname**。設定されていない場合は、ルート検出が実行されます。	オプション
SSO_DISABLE_SSL_CERTIFICATE_VALIDATION	**true** または **false** を選択して、Keycloak サーバー証明書の検証を有効または無効にします。SSO サーバーが自己署名証明書を生成する場合は、これを **true** に設定することを検討してください。	オプション
SSO_PASSWORD	Keycloak レルムと対話し、クライアントを作成および登録する権限を持つユーザーのパスワード。たとえば、**client-admin** などです。	True
SSO_REALM	アプリケーションクライアントを関連付けるための SSO レルム。たとえば、**saml-basic-auth** です。	オプション
SSO_SAML_CERTIFICATE_NAME	SAML クライアントキーストア内の秘密鍵と証明書のエイリアス。たとえば、**saml-app** です。	True
SSO_SAML_KEYSTORE	キーストアファイルの名前。たとえば、**keystore.jks** です。	True

環境変数	説明	必須
SSO_SAML_KEYSTORE_DIR	クライアントキーストアを含むディレクトリー。たとえば、**/etc/sso-saml-secret-volume** です。	True
SSO_SAML_KEYSTORE_PASSWORD	キーストアパスワードたとえば、**password** です。	True
SSO_SAML_LOGOUT_PAGE	ログアウトページ。たとえば、**simple-webapp-example** です。	True
SSO_SAML_VALIDATE_SIGNATURE	署名を検証する場合は **true** を、検証しない場合は **false** を指定します。デフォルトは true です。	オプション
SSO_SECURITY_DOMAIN	undertow および **ejb** サブシステムを保護するために使用するセキュリティードメインの名前。デフォルトは **keycloak** です。	オプション
SSO_TRUSTSTORE	サーバー証明書を含むトラストストアのファイル名。	オプション
SSO_TRUSTSTORE_CERTIFICATE_ALIAS	トラストストア内の証明書のエイリアス。	オプション
SSO_TRUSTSTORE_DIR	トラストストアを含むディレクトリー。	オプション
SSO_TRUSTSTORE_PASSWORD	トラストストアと証明書のパスワード。たとえば、**mykeystorepass** です。	オプション
SSO_URL	SSO サーバーの URL。たとえば、**<SSO server accessible route>** です。	True
SSO_USERNAME	Keycloak レルムと対話し、クライアントを作成および登録する権限を持つユーザーのユーザー名。たとえば、**client-admin** などです。	True

3.6.7. JBoss EAP サーバーのルート検出

JBoss EAP サーバーのルート検出機能を使用すると、サーバーのパフォーマンスを最適化し、指定した namespace でのルート設定を簡略化できます。この機能は、特に **HOSTNAME_HTTPS** 変数が指定されていない場合に、サーバーの効率を向上させ、よりスムーズな運用エクスペリエンスを実現する上で重要です。

HOSTNAME_HTTPS 変数が設定されていない場合、JBoss EAP サーバーは自動的にルート検出を試みます。ルート検出を有効にするには、必要な権限を作成する必要があります。

```
oc create role routeview --verb=list --resource=route -n YOUR_NAME_SPACE
oc policy add-role-to-user routeview system:serviceaccount:YOUR_NAME_SPACE:default --role-namespace=YOUR_NAME_SPACE -n YOUR_NAME_SPACE
```

3.6.8. 関連情報

- Red Hat build of Keycloak サーバー管理ガイド

3.7. 関連情報

- OpenShift Container Platform のスタートガイド

第4章 HELM チャートを使用した OPENSHIFT 上での JBOSS EAP アプリケーションのビルドおよびデプロイ

Helm は、OpenShift 上で JBoss EAP アプリケーションをビルド、デプロイ、保守できるようにするオープンソースのパッケージマネージャーです。JBoss EAP 8.0 では、OpenShift テンプレートの代わりに Helm チャートを使用します。

4.1. HELM チャートの使用例

JBoss EAP 8.0 で Helm チャートを使用すると、以下を行うことができます。

- OpenShift Source-to-Image (S2I) を使用して、Git リポジトリーでホストされている Maven プロジェクトからアプリケーションをビルドする。
- OpenShift クラスターとの緊密な統合 (TLS 設定、アプリケーションを公開するためのパブリックルートなど) を使用して、OpenShift にアプリケーションイメージをデプロイする。
- Helm チャートを使用してアプリケーションイメージをビルドし、JBoss EAP Operator を使用してイメージをデプロイする。
- その他の方法を使用して JBoss EAP のアプリケーションイメージをビルドし、Helm チャートを使用してアプリケーションイメージをデプロイする。

4.2. OPENSHIFT 上の JBOSS EAP に合わせた HELM チャートのカスタマイズ

アプリケーションの特定の設定を含む **YAML** ファイルを変更することで、JBoss EAP アプリケーションの Helm チャートをカスタマイズできます。

YAML ファイルには、次の 2 つの主要なセクションがあります。

- **build** 設定
- **deploy** 設定

YAML view で設定を選択すると、OpenShift 開発コンソールで直接 **Values ファイル** を編集し、更新した設定で Helm リリースをアップグレードできます。

関連情報

- JBoss EAP 8 の Helm チャート
- 値ファイルの例

4.3. S2I を使用した JBOSS EAP のプロビジョニング

アプリケーションの **pom.xml** の **eap-maven-plugin** を使用して、JBoss EAP サーバーをプロビジョニングします。

注記

build.s2i.featurePacks、**build.s2i.galleonLayers**、および **build.s2i.channels** フィールドは非推奨になりました。

4.4. HELM チャートを使用した JBOSS EAP アプリケーションのビルドとデプロイ

build 値と **deploy** 値を設定することで、Helms チャートを使用して JBoss EAP アプリケーションをビルドできます。**build** 設定には、アプリケーションコードをホストする Git リポジトリーへの URL を指定する必要があります。出力は、ビルドされたアプリケーションイメージを含む **ImageStreamTag** リソースです。

アプリケーションをデプロイするには、ビルドされたアプリケーションイメージを含む **ImageStreamTag** リソースを指定する必要があります。出力は、デプロイされたアプリケーションと、OpenShift 内外からアプリケーションにアクセスするために使用できるその他の関連リソースです。

前提条件

- OpenShift 開発コンソールにログインしている。
- Git リポジトリーでホストされている JBoss EAP アプリケーションがある。
- アプリケーションが Maven プロジェクトである。
- **org.jboss.eap.plugins:eap-maven-plugin** を使用して JBoss EAP 8.0 サーバーをプロビジョニングするようにアプリケーションを設定している。

手順

1. ソースリポジトリーからアプリケーションイメージをビルドします。

```
build:
  uri: <git repository URL of your application>
```

2. オプション: **build** セクションにシークレットを入力します。

```
build:
  sourceSecret: <name of secret login to your Git repository>
```

検証

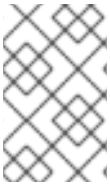
- アプリケーションが正常にデプロイされると、OpenShift 開発コンソールの Helm リリースの横にデプロイ済みバッジが表示されます。

関連情報

- Maven プラグインを使用した JBoss EAP サーバーのプロビジョニング

4.5. OPENSHIFT 開発コンソールを使用したアプリケーションイメージのビルド

OpenShift 開発コンソールで **build** セクションを設定することで、Helm チャートを使用して JBoss EAP アプリケーションイメージをビルドできます。



注記

別のメカニズムを使用してアプリケーションイメージをビルドした場合は、**build.enabled** フィールドを **false** に設定することで、Helm チャートのビルド部分をスキップできます。

重要

build.url フィールドには、Git リポジトリーを参照する Git URL を指定する必要があります。

関連情報

- JBoss EAP 8.0 の Helm チャート
- アプリケーションイメージのビルド

4.6. アプリケーションイメージのデプロイ

OpenShift 開発コンソールで **deploy** 設定を指定することで、Helm チャートを使用して JBoss EAP アプリケーションをデプロイできます。

注記

別のメカニズムを使用してアプリケーションイメージをビルドした場合は、**build.deploy** フィールドを **false** に設定することで、Helm チャートのデプロイ設定をスキップできます。

関連情報

- OpenShift Development Console Quickstarts
- Deploying the application image

4.6.1. Helm チャートの永続データストレージ用 OpenShift ボリューム

OpenShift ボリュームを使用すると、コンテナーで、クラウドストレージ、ネットワークファイルシステム (NFS)、ホストマシンなどのさまざまなソースからのデータを保存および共有できます。OpenShift のパッケージマネージャーである Helm チャートを使用すると、一貫性のある再現可能な方法でアプリケーションをデプロイできます。Helm チャートにボリュームマウントを追加すると、デプロイメント間でアプリケーションがデータを維持できるようになります。

4.6.2. Helm チャートを使用したボリュームのマウント

この手順では、JBoss EAP 8.0 で Helm チャートを使用して **secret** をボリュームとしてマウントする方法を説明します。さらに、これを使用して **ConfigMap** をマウントすることもできます。この手順を実行すると、アプリケーションがデータにアクセスして使用できるようになり、不正なアクセスや改ざんからデータを保護できます。

たとえば、**secret** をボリュームとしてマウントすると、シークレットに保存した機密データが、シークレットがマウントされているデプロイメントを実行している POD にファイルとして表示されます。

前提条件

- **secret** を作成している。たとえば、**keystore.jks** などのファイルを参照する **eap-app-secret** という名前のシークレットを作成します。
- コンテナーのファイルシステム内でシークレットをマウントする場所を特定している。たとえば、ディレクトリー **/etc/jgroups-encrypt-secre-secret-volume** は、**keystore.jks** などのシークレットファイルがマウントされる場所です。

手順

1. **deploy.volumes** フィールドで **volume** を指定し、使用するシークレットを設定します。ボリュームの **name** とシークレットの **secretName** を指定する必要があります。

```
volumes:
  - name: eap-jgroups-keystore-volume
    secret:
      secretName: eap-app-secret
```

2. デプロイメント設定の **deploy.volumeMounts** を使用して、ファイルシステムにボリュームをマウントします。

```
volumeMounts:
  - name: eap-jgroups-keystore-volume
    mountPath: /etc/jgroups-encrypt-secret-volume
    readOnly: true
```

Pod が起動すると、コンテナーが **keystore.jks** ファイルを **/etc/jgroups-encrypt-secret-volume/keystore.jks** にマウントします。

関連情報

- Volumes (Kubernetes ドキュメント)

第5章 環境変数とモデル式の解決

5.1. 前提条件

- オペレーティングシステムで環境変数を設定する方法について、ある程度の基本的な知識がある。
- OpenShift Container Platform で環境変数を設定するには、以下の前提条件を満たす必要がある。
  - すでに OpenShift をインストールし、OpenShift CLI ("oc") をセットアップしている。oc の詳細は、Getting Started with the OpenShift CLI を参照してください。
  - Helm チャートを使用してアプリケーションを OpenShift にデプロイした。Helm チャートの詳細は、Helm Charts for JBoss EAP を参照してください。

5.2. 管理モデル式を解決するための環境変数

管理モデル式を解決し、OpenShift Container Platform で JBoss EAP 8.0 サーバーを起動するには、環境変数を追加するか、管理コマンドラインインターフェイス (CLI) で Java システムプロパティーを設定します。両方を使用する場合、JBoss EAP は環境変数ではなく Java システムプロパティーを監視して使用し、管理モデル式を解決します。

システムプロパティーから環境変数へのマッピング
${my.example-expr} という管理式があるとします。JBoss EAP サーバーはこれを解決しようとするときに、**my.example-expr** という名前のシステムプロパティーをチェックします。

- サーバーがこのプロパティーを検出すると、その値を使用して式を解決します。
- このプロパティーが見つからない場合、サーバーは検索を続けます。

次に、サーバーがシステムプロパティー **my.example-expr** を見つけられないと仮定すると、サーバーは自動的に **my.example-expr** をすべて大文字に変更し、英数字以外のすべての文字をアンダースコア (_): **MY_EXAMPLE_EXPR** に置き換えます。次に、JBoss EAP はその名前の環境変数をチェックします。

- サーバーがこの変数を見つけると、その値を使用して式を解決します。
- この変数が見つからない場合、サーバーは検索を続けます。

ヒント

元の式が接頭辞 **env.** で始まる場合、JBoss EAP は接頭辞を削除して環境変数を解決し、環境変数名のみを検索します。たとえば、式 **env.example** の場合、JBoss EAP は **example** 環境変数を探します。

これらのチェックで元の式を解決するプロパティーまたは変数が見つからない場合、JBoss EAP は式にデフォルト値があるかどうかを探します。存在する場合、そのデフォルト値によって式が解決されます。そうでない場合、JBoss EAP は式を解決できません。

2 台のサーバーの例

1 つのサーバーで、JBoss EAP が次の管理リソースを定義するとします。**<socket-binding-group name="standard-sockets" default-interface="public" port-offset="${jboss.socket.binding.port-offset:0}">**設定ファイルを編集する代わりに、別のポートオフセットで 2 番目のサーバーを実行するには、次のいずれかを実行します。

- **jboss.socket.binding.port-offset** Java システムプロパティーを設定して、2 番目のサーバーで値を解決します: **./standalone.sh -Djboss.socket.binding.port-offset=100**。
- **JBOSS_SOCKET_BINDING_PORT_OFFSET** 環境変数を設定して、2 番目のサーバーの値を解決します: **JBOSS_SOCKET_BINDING_PORT_OFFSET=100 ./standalone.sh**。

5.3. OPENSHIFT CONTAINER PLATFORM での環境変数の設定

JBoss EAP 8.0 では、環境変数を設定して管理モデル式を解決できます。環境変数を使用して、OpenShift で実行している JBoss EAP サーバーの設定を適応させることもできます。

Pod テンプレートを使用するリソースに環境変数とオプションを設定します。

```
$ oc set env <object-selection> KEY_1=VAL_1 ... KEY_N=VAL_N [<set-env-options>] [<common-options>]
```

オプション	説明
-e, --env=<KEY>=<VAL>	環境変数のキーと値のペアを設定します。
--overwrite	既存の環境変数の更新を確認します。

注記

Pod テンプレートを使用する Kubernetes ワークロードリソースには、次のものがあります。

- **Deployment**
- **ReplicaSet**
- **StatefulSet**
- **DaemonSet**
- **Job**
- **CronJob**

環境変数を設定すると、JBoss EAP 管理コンソールは関連する Pod の詳細にそれらを表示するはずです。

関連情報

- OpenShift CLI について
- Red Hat JBoss Enterprise Application Platform 設定ガイド

5.4. 環境変数による管理属性のオーバーライド

Java システムプロパティーまたは環境変数を使用して、式で定義された管理属性を解決できることはわかっていますが、式を使用しない場合でも、他の属性を変更することもできます。

JBoss EAP サーバー設定をサーバー環境により簡単に適合させるために、設定ファイルを編集することなく、環境変数を使用して管理属性の値をオーバーライドできます。JBoss EAP バージョン 8.0 以降で利用できるこの機能は、以下の理由で役立ちます。

- JBoss EAP は、最も一般的な管理属性の式のみを提供します。式が定義されていない属性の値を変更できるようになりました。
- 一部の管理属性は、JBoss EAP サーバーを他のサービス (データベースなど) に接続しますが、その値を事前に知ることができないか、値を設定に保存することはできません。たとえば、データベース認証情報などです。環境変数を使用すると、JBoss EAP サーバーの実行中にそのような属性の設定を延期できます。

重要

この機能は、JBoss EAP バージョン 8.0 OpenShift ランタイムイメージ以降、デフォルトで有効になっています。他のプラットフォームで有効にするには、**WILDFLY_OVERRIDING_ENV_VARS** 環境変数を任意の値に設定する必要があります。たとえば、**WILDFLY_OVERRIDING_ENV_VARS=1 をエクスポート** します。

注記

type が **LIST**、**OBJECT**、または **PROPERTY** である管理属性をオーバーライドすることはできません。

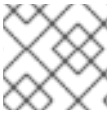
前提条件

- オーバーライドする管理属性を定義しておく必要があります。

手順

管理属性を環境変数でオーバーライドするには、次の手順を実行します。

1. 変更するリソースと属性のパスを特定します。たとえば、リソース **/subsystem=undertow/server=default-server/http-listener=default** の **proxy-address-forwarding** 属性の値を **true** に設定します。
2. 次のように、リソースアドレスと管理属性をマッピングして、この属性を上書きする環境変数の名前を作成します。
   a. リソースアドレスから最初のスラッシュ (/) を削除します。**/subsystem=undertow/server=default-server/http-listener=default** は、**subsystem=undertow/server=default-server/http-listener=default** になります。
   b. 2 つの下線 (__) と属性の名前を追加します。例: **subsystem=undertow/server=default-server/http-listener=default__proxy-address-forwarding**。
   c. 英数字以外のすべての文字をアンダースコア (_) に置き換え、コード行全体をすべて大文字にします: **SUBSYSTEM_UNDERTOW_SERVER_DEFAULT_SERVER_HTTP_LISTENER_DEFAULT__PROXY_ADDRESS_FORWARDING**。
3. 環境値を設定します: **SUBSYSTEM_UNDERTOW_SERVER_DEFAULT_SERVER_HTTP_LISTENER_DEFAULT__PROXY_ADDRESS_FORWARDING=true**。



注記

これらの値は例であり、実際の設定値に置き換える必要があります。

第6章 MAVEN プラグインを使用した JBOSS EAP サーバーのプロビジョニング

JBoss EAP Maven プラグインを使用すると、必要な機能を提供する Galleon レイヤーのみをサーバーに含めることで、要件に従ってサーバーを設定できます。

6.1. JBOSS EAP MAVEN プラグイン

JBoss EAP Maven プラグインは Galleon トリム機能を使用して、サーバーのサイズおよびメモリーフットプリントを削減します。JBoss EAP Maven プラグインは、サーバー設定をカスタマイズするための JBoss EAP CLI スクリプトファイルの実行をサポートします。CLI スクリプトには、サーバーを設定するための CLI コマンドのリストが含まれます。

Maven リポジトリーから最新の Maven プラグインバージョンを取得できます。これは、/ga/org/jboss/eap/plugins/eap-maven-plugin のインデックス にあります。Maven プロジェクトでは、**pom.xml** ファイルに JBoss EAP Maven プラグインの設定が含まれます。

JBoss EAP Maven プラグインは、サーバーをプロビジョニングし、Maven の実行中に WAR などのパッケージ化されたアプリケーションをプロビジョニングされたサーバーにデプロイします。アプリケーションがデプロイされるプロビジョニング済みサーバーは、**target/server** ディレクトリーにあります。JBoss EAP Maven プラグインでは、以下の機能も提供されます。

注記

target/server のサーバーはサポートされておらず、デバッグまたは開発目的でのみ使用できます。

- サーバー設定ファイルをカスタマイズするには、**org.jboss.eap:wildfly-galleon-pack** および **org.jboss.eap.cloud:eap-cloud-galleon-pack** Galleon **feature-pack** と、そのレイヤーの一部を使用します。
- CLI スクリプトコマンドをサーバーに適用します。
- **keystore** ファイルなど、サーバーインストールへの追加ファイルの追加をサポートします。

6.2. MAVEN を使用した JAKARTA EE 10 アプリケーションの作成

アクセスすると Hello World! を出力するアプリケーションを作成します。

前提条件

- JDK 17 がインストールされている。
- Maven 3.6 以降のバージョンがインストールされている。詳細は、Downloading Apache Maven を参照してください。

手順

1. Maven プロジェクトを設定します。

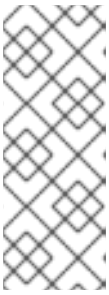
```
$ mvn archetype:generate \
-DgroupId=GROUP_ID \
-DartifactId=ARTIFACT_ID \
```

```
-DarchetypeGroupId=org.apache.maven.archetypes \
-DarchetypeArtifactId=maven-archetype-webapp \
-DinteractiveMode=false
```

GROUP_ID はプロジェクトの **groupId** で、ARTIFACT_ID はプロジェクトの **artifactId** です。

2. **jboss-eap-ee** BOM の Jakarta EE アーティファクトのバージョンを自動的に管理するように Maven を設定するには、プロジェクトの **pom.xml** ファイルの **<dependencyManagement>** セクションに BOM を追加します。以下に例を示します。

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.jboss.bom</groupId>
      <artifactId>jboss-eap-ee</artifactId>
      <version>8.0.0.GA-redhat-00009</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```



注記

- **<version>A.B.C-redhat-XXXXX</version> A.B.C** はリリース番号、**XXXXX** は JBoss EAP インスタンスのビルド番号です。JBoss EAP リリースのバージョンの詳細は、Red Hat Maven リポジトリーを参照してください。リリース番号とビルド番号は、すべての JBoss EAP リリースで利用できます。https://maven.repository.redhat.com/earlyaccess/all/org/jboss/bom/jboss-eap-ee/.

3. 以下の例のように、BOM によって管理されるサーブレット API アーティファクトをプロジェクトの **pom.xml** ファイルの **<dependencies>** セクションに追加します。

```
<dependency>
    <groupId>jakarta.servlet</groupId>
    <artifactId>jakarta.servlet-api</artifactId>
</dependency>
```

4. 以下の内容で Java ファイル **TestServlet.java** を作成し、ファイルを **APPLICATION_ROOT/src/main/java/com/example/simple/** ディレクトリーに保存します。

```
package com.example.simple;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.io.PrintWriter;
@WebServlet(urlPatterns = "/hello")
public class TestServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
IOException {
```

```
        PrintWriter writer = resp.getWriter();
        writer.println("Hello World!");
        writer.close();
    }
}
```

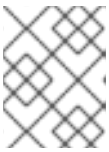
このアプリケーションを JBoss EAP にデプロイするか、このアプリケーションを更新してパッケージ化し、Maven プラグインを使用してカスタムプロビジョニングされた JBoss EAP サーバーにデプロイできるようになりました。

6.3. MAVEN プラグインを使用した JBOSS EAP サーバーのプロビジョニング

アプリケーションの **pom.xml** を更新してパッケージ化し、Maven プラグインを使用してカスタムプロビジョニングされた JBoss EAP サーバーにデプロイします。その後、カスタムプロビジョニングされた JBoss EAP サーバーで実行されているアプリケーションを OpenShift にデプロイできます。

前提条件

- JBoss EAP Maven プラグインと JBoss EAP Maven アーティファクトが、ローカルまたはリモートの Maven リポジトリーからアクセスできる。
- JDK 17 がインストールされている。
- Maven がインストールされている。詳細は、Downloading Apache Maven を参照してください。



注記

JDK 17 と Maven 3.8.5 またはそれ以前の Maven バージョンを使用している場合は、最新の Maven WAR プラグインを使用してください。

- Jakarta EE 10 アプリケーション用の Maven プロジェクトを作成している。詳細は、Maven を使用した Jakarta EE 10 アプリケーションの作成 を参照してください。

手順

1. 以下のコンテンツを **pom.xml** ファイルに追加して、リモートリポジトリーから JBoss EAP BOM および JBoss EAP Maven プラグインを取得するように Maven を設定します。

```
<repositories>
    <repository>
        <id>jboss</id>
        <url>https://maven.repository.redhat.com/ga/</url>
        <snapshots>
            <enabled>false</enabled>
        </snapshots>
    </repository>
</repositories>
<pluginRepositories>
    <pluginRepository>
        <id>jboss</id>
        <url>https://maven.repository.redhat.com/ga/</url>
        <snapshots>
```

```
            <enabled>false</enabled>
        </snapshots>
    </pluginRepository>
</pluginRepositories>
```

2. 以下の内容を **pom.xml** ファイルの **<build>** 要素に追加します。必ず JBoss EAP Maven プラグインの最新バージョンを指定してください。以下に例を示します。

```
<plugins>
    <plugin>
        <groupId>org.jboss.eap.plugins</groupId>
        <artifactId>eap-maven-plugin</artifactId>
        <version>1.0.0.Final-redhat-00014</version> (1)
        <configuration>
            <channels>
                <channel>
                    <manifest>
                        <groupId>org.jboss.eap.channels</groupId> (2)
                        <artifactId>eap-8.0</artifactId>
                    </manifest>
                </channel>
            </channels>
            <feature-packs>
                <feature-pack>
                    <location>org.jboss.eap:wildfly-ee-galleon-pack</location> (3)
                </feature-pack>
                <feature-pack>
                    <location>org.jboss.eap.cloud:eap-cloud-galleon-pack</location> (4)
                </feature-pack>
            </feature-packs>
            <layers>
                <layer>cloud-server</layer> (5)
            </layers>
            <runtime-name>ROOT.war</runtime-name> (6)
        </configuration>
        <executions>
            <execution>
                <goals>
                    <goal>package</goal> (7)
                </goals>
            </execution>
        </executions>
    </plugin>
</plugins>
```

(1) **<version>1.0.0.Final-redhat-00014</version>** は、JBoss EAP Maven プラグインのバージョンの例です。JBoss EAP Maven プラグインのリリースの詳細は、Red Hat Maven リポジトリー (https://maven.repository.redhat.com/earlyaccess/all/org/jboss/eap/plugins/eap-maven-plugin/) を参照してください。

(2) これは、JBoss EAP サーバーアーティファクトが定義されている JBoss EAP 8.0 チャネルを指定します。

(3) この機能パックのバージョンは、JBoss EAP チャネルから取得できます。Galleon

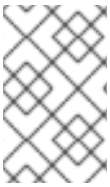
feature-pack には、トリムされた JBoss EAP サーバーをプロビジョニングする **cloud-server** などの Galleon レイヤーが含まれています。

4. この機能パックは、クラウドのサーバー Galleon レイヤーを調整します。OpenShift 用のアプリケーションをビルドするには、この機能パックを使用する必要があります。
5. この Galleon レイヤーは、クラウドで JBoss EAP アプリケーションを実行するときに必要な機能を備えたサーバーをプロビジョニングします。
6. この設定オプションを使用すると、デプロイを HTTP ルートコンテキストに登録できます。
7. このプラグインゴールを使用すると、サーバーのプロビジョニング、アプリケーションのデプロイ、カスタム設定の CLI スクリプトの適用、カスタムコンテンツのサーバーインストールへのコピーを行うことができます。

3. アプリケーションをパッケージ化します。

```
$ mvn package
```

ディレクトリー **target/server** には、デバッグまたは開発目的ですぐに使用できるサーバーとアプリケーションが含まれています。JBoss EAP S2I ビルドコンテキストでは、JBoss EAP maven-plugin によってプロビジョニングされたサーバーが **/opt/server** の場所にある JBoss EAP イメージにインストールされます。詳細は、Source-to-Image (S2I) を使用した OpenShift でのアプリケーションイメージの作成 を参照してください。



注記

デバッグを有効にして **mvn package** コマンドを使用する場合 (**-X** オプション)、スキーマ検証中にデバッグログが過剰に記録されるのを防ぐために、プロパティー **-Dorg.slf4j.simpleLogger.log.com.networknt.schema=off** を追加してください。

検証

- 生成されたサーバー設定ファイル **target/server/standalone/configuration/standalone.xml** を確認できます。このファイルには、プロビジョニングされたサブシステムとアプリケーションのデプロイメントが含まれています。

デプロイメントを含む JBoss EAP サーバーがプロビジョニングされました。

関連情報

- 利用可能な JBoss EAP レイヤー
- Use the Maven Repository
- Introduction to the Standard Directory Layout in the Apache Maven documentation

6.4. GALLEON プロビジョニングファイル

プロビジョニングファイルは、**galeon** サブディレクトリーに格納できる **provisioning.xml** という名前の XML ファイルです。JBoss EAP Maven プラグインで機能パックとレイヤーを設定する代わりに、これを使用できます。**provisioning.xml** ファイルを設定して、プロビジョニングプロセスを微調整することもできます。

以下のコードは、**cloud-server** レイヤーに基づいて JBoss EAP サーバーをプロビジョニングするために使用できるプロビジョニングファイルの内容を示しています。

注記

JBoss EAP 機能パックにはバージョンがありません。バージョンは Maven プラグインで設定されたチャネルから取得されます。

```
<?xml version="1.0" ?>
<installation xmlns="urn:jboss:galleon:provisioning:3.0">
    <feature-pack location="org.jboss.eap:wildfly-ee-galleon-pack:">1
        <default-configs inherit="false"/>2
        <packages inherit="false"/>3
    </feature-pack>
    <feature-pack location="org.jboss.eap.cloud:eap-cloud-galleon-pack:
">4
        <default-configs inherit="false"/>
        <packages inherit="false"/>
    </feature-pack>
    <config model="standalone" name="standalone.xml">5
        <layers>
            <include name="cloud-server"/>
        </layers>
    </config>
    <options>6
        <option name="optional-packages" value="passive+"/>
    </options>
</installation>
```

1. この要素は、JBoss EAP チャネルから取得した JBoss EAP 機能パックをプロビジョニングするようにプロビジョニングプロセスに指示します。
2. この要素は、デフォルト設定を除外するようにプロビジョニングプロセスに指示します。**standalone.xml** や **standalone-ha.xml** など、JBoss EAP サーバーのインストールでデフォルト設定を取得できます。JBoss EAP Maven プラグインから JBoss EAP サーバーをプロビジョニングする場合、設定された Galleon ユーザーに基づいて単一のサーバー設定を生成します。オプションを **false** に設定すると、追加のサーバー設定が生成されなくなります。**inherit=true** の設定は、**default-configs** と **packages** の両方でサポートされていません。
3. この要素は、デフォルトパッケージを除外するようにプロビジョニングプロセスに指示します。
4. この要素は、JBoss EAP クラウド機能パックをプロビジョニングするようにプロビジョニングプロセスに指示します。子要素は、デフォルトの設定およびデフォルトパッケージを除外するようプロセスに指示します。
5. この要素は、カスタムスタンドアロン設定を作成するようにプロビジョニングプロセスに指示します。この設定には、JBoss EAP 機能パックで定義され、JBoss EAP クラウド機能パックによって OpenShift 用に調整された **cloud-server** 基本レイヤーが含まれます。
6. この要素は、JBoss EAP モジュールのプロビジョニングを最適化するようプロビジョニングプロセスに指示します。

6.5. MAVEN プラグインの設定属性

次の設定パラメーターのリストを設定することにより、**eap-maven-plugin** Maven プラグインを設定できます。

表6.1 Maven プラグインの設定属性

名前	型	説明
channels	List	チャネル YAML ファイルのリファレンスリスト。チャネルファイルには、JBoss EAP サーバーアーティファクトのバージョンが含まれています。チャネル YAML ファイルを識別する方法は 2 つあります。 ● チャネル分類子を使用してチャネル YAML ファイルアーティファクトを Maven リポジトリーにデプロイすると、その Maven 座標 (**groupId**、**artifactId**、およびオプションのバージョン) を使用して識別できます。バージョンが設定されていない場合は、最新のチャネルバージョンが使用されます。以下に例を示します。 `<channels> <channel> <manifest> <groupId>org.jboss.eap.channels</groupId> <artifactId>eap-8.0</artifactId> </manifest> </channel> </channels>` ● URL を使用してチャネル YAML ファイルを取得できます。以下に例を示します。 `<channels> <channel> <manifest> <url>file:///foo/my-manifest.yaml</url> </manifest> </channel> </channels>`
excluded-layers	List	除外する Galleon レイヤーのリスト。**feature-pack-location** または機能パックが設定されている場合に使用できます。システムプロパティー **wildfly.provisioning.layers.excluded** を使用して、除外するレイヤーのコンマ区切りリストを指定します。
extra-server-content-dirs	List	コンテンツがプロビジョニングされたサーバーにコピーされるディレクトリーのリスト。ディレクトリーへの絶対パスまたは相対パスのいずれかを使用できます。相対パスは、プロジェクトのベースディレクトリーからの相対パスである必要があります。

名前	型	説明
feature-packs	List	レイヤーと組み合わせることができる、インストールする機能パック設定のリスト。システムプロパティー **wildfly.provisioning.feature-packs** を使用して、機能パックのコンマ区切りリストを提供します。
filename	String	デプロイするアプリケーションのファイル名。デフォルト値は **${project.build.finalName}.${project.packaging}** です。例外として、**ejb** パッケージングの場合は拡張子が **.jar** になります。たとえば、war パッケージング時の **$[project.packaging]** の値は **war** であり、ejb パッケージング時の **$[project.packaging]** の値は **ejb** ですが、これは有効な jar 拡張子ではありません。このような場合には、**.jar** 拡張子が必要です。

galleon-options	Map	サーバーのプロビジョニング時に、特定の Galleon オプションを設定できます。同じ Maven セッションで多数のサーバーをビルドしている場合、**jboss-fork-embedded** オプションを **true** に設定して、Galeon プロビジョニングと CLI スクリプトの実行をフォークする必要があります。以下に例を示します。
		
		`<galleon-options>`
		`  <jboss-fork-embedded>true</jboss-fork-embedded>`
		`</galleon-options>`

layers	List	プロビジョニングする Galleon レイヤーのリスト。**feature-pack-location** または機能パックが設定されている場合に使用できます。システムプロパティー **wildfly.provisioning.layers** を使用して、レイヤーのコンマ区切りリストを提供します。
layers-configuration-file-name	String	レイヤーから生成された設定ファイルの名前。デフォルト値は **standalone.xml** です。レイヤーが設定されていない場合、このパラメーターは設定できません。
log-provisioning-time	boolean	プロビジョニング終了時にプロビジョニング時間をログに記録するかどうかを指定します。デフォルト値は **false** です。
name	String	デプロイメントに使用される名前。

名前	型	説明
offline-provisioning	boolean	プラグインがアーティファクトを解決するときにオフラインモードを使用するかどうかを指定します。オフラインモードでは、プラグインはローカルの Maven リポジトリーを使用してアーティファクトを解決します。デフォルト値は **false** です。
overwrite-provisioned-server	boolean	**provisioningDir** から参照されている既存のサーバーを削除し、新しいサーバーをプロビジョニングする場合は、**true** に設定します。そうでない場合は、**false** に設定します。デフォルト値は **false** です。

packaging-scripts	List	実行する CLI スクリプトとコマンドのリスト。スクリプトファイルが絶対ファイルでない場合は、プロジェクトのベースディレクトリーの相対ファイルである必要があります。次の方法で CLI の実行を設定します。
		
		`<packaging-scripts>`
		`  <packaging-script>`
		`  <scripts>`
		`  <script>../scripts/script1.cli</script>`
		`  </scripts>`
		`  <commands>`
		`  <command>/system-property=foo:add(value=bar)</command>`
		`  </commands>`
		`  <properties-files>`
		`  <property-file>my-properties.properties</property-file>`
		`  </properties-files>`
		`  <java-opts>`
		`  <java-opt>-Xmx256m</java-opt>`
		`  </java-opts>`
		`  <!-- Expressions resolved during server execution -->`
		`  <resolve-expressions>false</resolve-expressions>`
		`  </packaging-script>`
		`</packaging-scripts>`

provisioning-dir	String	サーバーをプロビジョニングするディレクトリーへのパス。絶対パスまたは **buildDir** への相対パスにできます。デフォルトでは、サーバーは **target/server** ディレクトリーにプロビジョニングされます。デフォルト値は **server** です。

名前	型	説明
provisioning-file	File	使用する **provisioning.xml** ファイルへのパス。機能パック設定アイテムとレイヤー設定アイテムが設定されている場合は使用できません。プロビジョニングファイルのパスが絶対パスでない場合は、プロジェクトのベースディレクトリーへの相対パスにする必要があります。デフォルト値は **${project.basedir}/galleon/provisioning.xml** です。
record-provisioning-state	boolean	プロビジョニングの状態を **.galleon** ディレクトリーに記録するかどうかを指定します。デフォルト値は **false** です。
runtime-name	String	デプロイメントのランタイム名。デフォルト値は、**myapp.war** などのデプロイメントファイル名です。この引数を **ROOT.war** に設定して、HTTP ルートコンテキストに登録されたデプロイメントを取得できます。
server-config	String	デプロイ時に使用するサーバー設定の名前。**layers-configuration-file-name** が設定されている場合、デプロイメントは **layers-configuration-file-name** から参照される設定内にデプロイされます。デフォルト値は **standalone.xml** です。
skip	boolean	ゴールをスキップする場合は、**true** に設定します。そうでない場合は、**false** に設定します。デフォルト値は **false** です。
stdout	String	作成された CLI プロセスの **stdout** および **stderr** がどのように処理されるかを示します。値が定義されている場合、その値が none でない限り **stderr** は **stdout** にリダイレクトされます。デフォルトでは、**stdout** および **stderr** ストリームは現在のプロセスから継承されます。次のいずれかのオプションから設定を変更できます。 ● **None** は、**stderr** と **stdout** を使用しないことを示します。 ● **System.out** または **System.err** は、現在のプロセスにリダイレクトします。 ● その他の値はファイルへのパスと見なされ、**stdout** と **stderr** がそれに書き込まれます。

6.6. JBOSS EAP 8.0 の EAP-DATASOURCES-GALLEON-PACK のサポートを有効にする方法

eap-data-sources-galleon-pack Galleon 機能パックを使用すると、データベースに接続できる JBoss EAP 8.0 サーバーをプロビジョニングできます。

注記

すべてのデータベースがサポートされているわけではありません。

さらに、この機能パックは、JBoss EAP 8.0 Galleon 機能パックとともにプロビジョニングできるさまざまなデータベース用の JDBC ドライバーとデータソースを提供します。この機能パックで定義されている Galleon レイヤーは、デコレーターレイヤーです。デコレーターレイヤーは、JBoss EAP のベースレイヤーに加えてプロビジョニングする必要があるレイヤーです。

重要

datasources-web-server ベースレイヤーは、機能パックで定義されている Galleon レイヤーをプロビジョニングするときに使用する最小限のベースレイヤーです。

関連情報

- JBoss EAP 8.0 定義のレイヤーに関するドキュメント

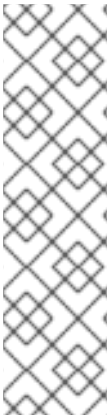
6.7. サポートされているドライバーとデータソース

Galleon 機能パックは、サポートするデータベースごとに、他のレイヤーを基盤とする Galleon レイヤーを提供します。これらは次のとおりです。

- **postgresql-driver**
- **postgresql-datasource**
- **mssqlserver-datasource**
- **mssqlserver-driver**
- **oracle-datasource**
- **oracle-driver**

表6.2 サポートされているドライバーとデータソース

レイヤー	説明
postgresql-driver	ドライバーの JBoss EAP モジュールをインストールし、サーバー設定のデータソースサブシステムにドライバーリソースを追加します。
postgresql-datasource	**postgresql-driver** Galleon レイヤーを基盤としており、データソースを追加します。



注記

- 機能パックには特定のドライバーのバージョンは含まれていません。サーバーをプロビジョニングする前に、ドライバーのバージョンを指定する必要があります。

  例:

  POSTGRESQL_DRIVER_VERSION="42.2.19"

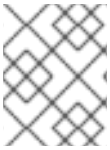
- ドライバー固有の環境変数は、各ドライバーのドキュメント内で定義されています。

関連情報

- Microsoft SQL Server ドライバーとデータソースを設定するための環境変数
- Oracle ドライバーとデータソースを設定するための環境変数
- PostgreSQL ドライバーとデータソースを設定するための環境変数

6.8. JBOSS EAP MAVEN プラグインを使用して JDBC ドライバーとデータソースを備えたサーバーをプロビジョニングする

Galleon 機能パックを使用して、OpenShift で JBoss EAP サーバーをプロビジョニングできます。



注記

この手順では、JBoss EAP 8.0 向けに OpenShift で JBoss EAP サーバーをプロビジョニングする方法のみを説明します。

前提条件

- すでに OpenShift をインストールし、OpenShift CLI ("oc") をセットアップしている。詳細は、Getting Started with the OpenShift CLI を参照してください。
- JBoss EAP Maven プラグインの使用方法に関する基本的な知識を持っている。詳細は、JBoss EAP Maven プラグイン を参照してください。

手順

- データソース機能パックの Maven コーディネート (GroupId および artifactId) を JBoss EAP Maven プラグイン設定に追加します。

```
<channels>
  <channel>
    <groupId>org.jboss.eap.channels</groupId>
    <artifactId>eap-8.0</artifactId>
  </channel>
</channels>
<feature-packs>
  <feature-pack>
    <location>org.jboss.eap:wildfly-ee-galleon-pack</location>
  </feature-pack>
  <feature-pack>
```

```
    <location>org.jboss.eap.cloud:eap-cloud-galleon-pack</location>
  </feature-pack>
  <feature-pack>
    <location>org.jboss.eap:eap-datasources-galleon-pack</location>
  </feature-pack>
</feature-packs>
<layers>
  <!-- Base layer -->
  <layer>jaxrs-server</layer>
    <!-- The postgresql datasource layer -->
  <layer>postgresql-datasource</layer>
</layers>
```

関連情報

- todo-backend クイックスタート用の JBoss EAP 8 Beta Maven プラグイン設定

第7章 JBOSS EAP サーバーとアプリケーションの設定

JBoss EAP for OpenShift のイメージは、Java アプリケーションとの基本的な使用に対して事前設定されています。しかし、JBoss EAP インスタンスをイメージ内部で設定できます。推奨される方法は、OpenShift S2I プロセスを使用し、Helm チャートで環境変数を設定して JVM を調整することです。

重要

コンテナーが再起動または終了すると、実行中のコンテナーで変更された設定内容はすべて失われます。

これには、**add-user.sh** や管理 CLI などの、従来の JBoss EAP インストールに含まれるスクリプトを使用して変更された設定が含まれます。

JBoss EAP for OpenShift イメージ内の JBoss EAP インスタンスに設定変更を加えるには、OpenShift S2I プロセスを環境変数とともに使用することを強く推奨します。

7.1. JVM のデフォルトメモリー設定

以下の環境変数を使用して、自動的に計算された JVM 設定を変更できます。これらの変数は、デフォルトメモリーサイズが自動的に算出される場合にのみ使用されることに注意してください (有効なコンテナーのメモリー制限が定義されているとき)。

環境変数	説明
JAVA_INITIAL_MEM_RATIO	この環境変数は非推奨になりました。JVM 引数 **-XX:InitialRAMPercentage** に対応します。これはデフォルトでは指定されておらず、将来のリリースで削除される予定です。**--XX:InitialRAMPercentage** を **JAVA_OPTS** で直接指定する必要があります。  **注記** 自動計算を無効にするために **JAVA_INITIAL_MEM_RATIO=0** を設定する必要はなくなりました。この環境変数にはデフォルト値が提供されていないためです。
JAVA_MAX_MEM_RATIO	**-XX:MaxRAMPercentage** JVM オプションを設定するための環境変数。最大ヒープサイズを、Java VM で使用可能な合計メモリーのパーセンテージとして設定します。デフォルト値は 80% です。**JAVA_MAX_MEM_RATIO=0** を設定すると、このデフォルト値が無効になります。
JAVA_OPTS	JVM に追加のオプションを提供するための環境変数 (例: **JAVA_OPTS=-Xms512m -Xmx1024m)**  **注記** **-Xms** に値を設定すると、**-XX:InitialRAMPercentage** オプションは無視されます。**-Xmx** に値を設定すると、**-XX:MaxRAMPercentage** オプションは無視されます。

環境変数	説明
JAVA_MAX_INITIAL_MEM	この環境変数は非推奨になりました。**JAVA_OPTS** を使用して `-Xms` オプションを指定します。たとえば、**JAVA_OPTS=-Xms256m です。**

7.2. JVM ガベージコレクションの設定

OpenShift の EAP イメージには、コレクションとガベージコレクションロギングの両方の設定が含まれます。

ガベージコレクションの設定

-XX:+UseParallelGC -XX:MinHeapFreeRatio=10 -XX:MaxHeapFreeRatio=20 -XX:GCTimeRatio=4 -XX:AdaptiveSizePolicyWeight=90 -XX:+ExitOnOutOfMemoryError

ガベージコレクションのログ設定

-Xlog:gc*:file=/opt/server/standalone/log/gc.log:time,uptimemillis:filecount=5,filesize=3M

7.3. JVM 環境変数

これらの環境変数を使用して、EAP for OpenShift イメージで JVM を設定します。

表7.1 JVM 環境変数

変数名	例:	デフォルト値	JVM の設定	説明

変数名	例:	デフォルト値	JVM の設定	説明
JAVA_OPTS	-verbose:class	デフォルトなし	Multiple	**java** コマンドに渡す JVM オプション。 **JAVA_OPTS_APPEND** を使用して追加の JVM 設定を設定します。**JAVA_OPTS** を使用すると、一部の未設定のデフォルト値がサーバー JVM 設定に追加されません。これらの設定は、明示的に追加する必要があります。 **JAVA_OPTS** を使用すると、コンテナースクリプトによりデフォルトで追加された特定の設定が無効になります。無効な設定には以下が含まれます。 ● -XX:MetaspaceSize=96M ● -Djava.net.preferIPv4Stack=true ● -Djboss.modules.system.pkgs=jdk.nashorn.api,com.sun.crypto.provider ● -Djava.awt.headless=true **JAVA_OPTS** を使用して追加設定を行う場合は、これらのデフォルトを追加します。
JAVA_OPTS_APPEND	-Dsome.property=value	デフォルトなし	Multiple	**JAVA_OPTS** で生成されたオプションに追加するユーザー指定の Java オプション。
JAVA_MAX_MEM_RATIO	50	80	-Xmx	**-Xmx** オプションが **JAVA_OPTS** に指定されていない場合には、この変数を使用します。この変数の値は、コンテナーの制限に基づいてデフォルトの最大ヒープメモリーサイズを算出するために使用されます。この変数がメモリー制限なしで、コンテナー内で使用される場合、この変数の効果はありません。この変数がメモリー制約のあるコンテナーで使用されると、**-Xmx** の値はコンテナーで使用できるメモリーの指定の比率に設定されます。デフォルト値の 50 は、利用可能なメモリーの 50% が上限として使用されることを意味します。最大メモリーの計算を省略するには、この変数の値を 0 に設定します。**JAVA_OPTS** には、**-Xmx** オプションは追加されません。

変数名	例:	デフォルト値	JVM の設定	説明
JAVA_INITIAL_MEM_RATIO	25	-Xms	-Xms	この変数は、**-Xms** オプションが **JAVA_OPTS** で指定されていない場合に使用します。この変数の値は、最大ヒープメモリーサイズを基にしたデフォルトの初期ヒープメモリーサイズの算出に使用されます。この変数がメモリー制限なしで、コンテナー内で使用される場合、この変数の効果はありません。この変数がメモリー制約のあるコンテナーで使用されている場合、**-Xms** の値が **Xmx** メモリーの指定比に設定されます。デフォルト値の **25** は、初期ヒープサイズとして最大メモリーの 25% が使用されることを意味します。初期メモリーの計算を省略するには、この変数の値を **0** に設定します。**JAVA_OPTS** には **-Xms** オプションは追加されません。
JAVA_MAX_INITIAL_MEM	4096	4096	-Xms	**JAVA_MAX_INITIAL_MEM** 環境変数は非推奨になりました。**JAVA_OPTS** を使用して -Xms オプションを指定してください。例: JAVA_OPTS=-Xms256m
JAVA_DIAGNOSTICS	true	false (無効)	**-Xlog:gc:utctime -XX:NativeMemoryTracking=summary**	この変数の値を **true** に設定すると、イベントの発生時に、標準出力に診断情報が含まれます。**JAVA_DIAGNOSTICS** がすでに **true** として定義されている環境で、この値が **true** に定義されていると、診断が依然として含まれます。
DEBUG	true	false	-agentlib:jdwp=transport=dt_socket,address=$DEBUG_PORT,server=y,suspend=n	リモートデバッグを有効にします。
DEBUG_PORT	8787	8787	-agentlib:jdwp=transport=dt_socket,address=$DEBUG_PORT,server=y,suspend=n	デバッグに使用するポートを指定します。
GC_MIN_HEAP_FREE_RATIO	20	10	-XX:MinHeapFreeRatio	拡大を回避するためのガベージコレクション後のヒープ解放の最小パーセンテージ。

変数名	例:	デフォルト値	JVM の設定	説明
GC_MAX_HEAP_FREE_RATIO	40	20	-XX:MaxHeapFreeRatio	縮小を回避するためのガベージコレクション後のヒープ解放の最大パーセンテージ。
GC_TIME_RATIO	4	4	-XX:GCTimeRatio	ガベージコレクションで費やした時間と、それ以外で費やされる時間の比率を指定します (アプリケーション実行にかかった時間など)。
GC_ADAPTIVE_SIZE_POLICY_WEIGHT	90	90	-XX:AdaptiveSizePolicyWeight	現在のガベージコレクション時間と以前のガベージコレクション時間に指定される重み。
GC_METASPACE_SIZE	20	96	-XX:MetaspaceSize	初期メタスペースのサイズ。
GC_MAX_METASPACE_SIZE	100	デフォルトなし	-XX:MaxMetaspaceSize	最大メタスペースサイズ。
GC_CONTAINER_OPTIONS	-XX:+UseG1GC	-XX:-UseParallelGC	-XX:-UseParallelGC	使用する Java ガベージコレクションを指定します。変数の値は、Java ランタイム環境 (JRE) コマンド行オプションを使用して指定されます。指定された JRE コマンドは、デフォルトをオーバーライドします。

以下の環境変数が非推奨になりました。

- **JAVA_OPTIONS**: **JAVA_OPTS** を使用します。
- **INITIAL_HEAP_PERCENT**: **JAVA_INITIAL_MEM_RATIO** を使用します。
- **CONTAINER_HEAP_PERCENT**: **JAVA_MAX_MEM_RATIO** を使用します。

7.4. デフォルトデータソース

データソース **ExampleDS** は、JBoss EAP 8.0 では使用できません。

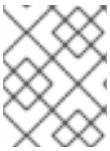
一部のクイックスタートには、以下のデータソースが必要です。

- **cmt**
- **thread-racing**

お客様が開発したアプリケーションでも、**ExampleDS** のデータソースが必要になる場合があります。

デフォルトのデータソースが必要な場合は、**ENABLE_GENERATE_DEFAULT_DATASOURCE** 環境変数を使用して、JBoss EAP サーバーのプロビジョニング時にこれを含めます。

```
ENABLE_GENERATE_DEFAULT_DATASOURCE=true
```



注記

この環境変数は、**cloud-default-config** Galleon レイヤーが使用されている場合にのみ機能します。

第8章 JBOSS EAP FOR OPENSHIFT の機能のトリム

サーバーの機能をトリムすると、プロビジョニングされたサーバーのセキュリティーリスクを軽減したり、マイクロサービスコンテナーにより適したサーバーにするためにメモリーフットプリントを削減したりできます。

JBoss EAP を含むイメージをビルドする際に、イメージに含める JBoss EAP の機能とサブシステムを制御できます。これを行うには、Source-to-Image (S2I) ビルドプロセス中、新しいアプリケーションを作成するときに JBoss EAP Maven プラグインを使用します。詳細は、Maven プラグインを使用した JBoss EAP サーバーのプロビジョニング を参照してください。

注記

S2I ビルドプロセス中に、JBoss EAP Maven プラグインの代わりに次の環境変数を使用できます。

- GALLEON_PROVISION_FEATURE_PACKS
- GALLEON_PROVISION_LAYERS
- GALLEON_PROVISION_CHANNELS

8.1. 利用可能な JBOSS EAP レイヤー

Red Hat は、OpenShift での JBoss EAP サーバーのプロビジョニングをカスタマイズできる基本レイヤーとデコレーターレイヤーを提供します。ベースレイヤーはコア機能を提供し、デコレーターレイヤーはベースレイヤーを強化します。

プロビジョニング層では、以下の Jakarta EE 仕様はサポートされません。

- Jakarta Server Faces 2.3
- Jakarta Enterprise Beans 3.2
- Jakarta XML Web Services 2.3

8.1.1. ベースレイヤー

各ベースレイヤーには、典型的なサーバーユーザーケースのコア機能が含まれています。

datasources-web-server
このレイヤーには、サーブレットコンテナーが含まれ、データソースを設定する機能が含まれます。

以下は、**datasources-web-server** にデフォルトで含まれている JBoss EAP サブシステムです。

- **core-management**
- **datasources**
- **deployment-scanner**
- **ee**
- **elytron**
- **io**

- **jca**
- **jmx**
- **logging**
- **命名**
- **request-controller**
- **security-manager**
- **transactions**
- **undertow**

このレイヤーでは、以下の Jakarta EE 仕様がサポートされます。

- Jakarta JSON Processing 1.1
- Jakarta JSON Binding 1.0
- Jakarta Servlet 4.0
- Jakarta Expression Language 3.0
- Jakarta Server Pages 2.3
- Jakarta Standard Tag Library 1.2
- Jakarta Concurrency 1.1
- Jakarta Annotations 1.3
- Jakarta XML Binding 2.3
- Jakarta Debugging Support for Other Languages 1.0
- Jakarta Transactions 1.3
- Jakarta Connectors 1.7

jaxrs-server

このレイヤーは、以下の JBoss EAP サブシステムを使用して **datasources-web-server** レイヤーを強化します。

- **jaxrs**
- **weld**
- **jpa**

また、このレイヤーは、ローカルキャッシュを備えた Infinispan ベースのセカンドレベルのエンティティーをコンテナーに追加します。

以下の Jakarta EE 仕様は、**datasources-web-server** レイヤーでサポートされるものに加え、このレイヤーでサポートされています。

- Jakarta Contexts and Dependency Injection 2.0
- Jakarta Bean Validation 2.0
- Jakarta Interceptors 1.2
- Jakarta RESTful Web Services 2.1
- Jakarta Persistence 2.2

cloud-server
このレイヤーは、以下の JBoss EAP サブシステムを使用して **jaxrs-server** レイヤーを強化します。

- **resource-adapters**
- **messaging-activemq** (組み込みメッセージではなく、リモートブラオーカーメッセージング)

このレイヤーは、以下の observability 機能も **jaxrs-server** レイヤーに追加します。

- ネイティブヘルス
- ネイティブメトリクス

以下の Jakarta EE 仕様は、**jaxrs-server** レイヤーでサポートされるものに加え、このレイヤーでサポートされています。

- Jakarta Security 1.0

cloud-default-config
このレイヤーは、**standalone-ha.xml** に基づくサーバー設定でサーバーをプロビジョニングし、サブシステム設定の **messaging-activemq** を含みます。逆に、**modcluster** と **core-management** サブシステムの設定は含まれていません。これはクラウドで使用するように設定されています。さらに、すべての JBoss EAP サーバー JBoss モジュールがインストールされます。

ee-core-profile-server
ee-core-profile-server レイヤーは、Jakarta EE 10 Core Profile を使用してサーバーをプロビジョニングします。Core Profile は、コア JBoss EAP サーバー機能と Jakarta EE API の両方を提供する小さく軽量なプロファイルをユーザーに提供します。**ee-core-profile-server** レイヤーは、クラウドネイティブアプリケーションやマイクロサービスなどの小規模なランタイムに最適です。

8.1.2. デコレーターレイヤー

デコレーターレイヤーは単独で使用されません。ベースレイヤーで 1 つ以上のデコレーターレイヤーを設定することで、追加機能を利用できます。

observability
このデコレーターレイヤーは、プロビジョニングしたサーバーに以下の observability 機能を追加します。

- ネイティブヘルス
- ネイティブメトリクス

注記

このレイヤーは **cloud-server** レイヤーに組み込まれています。このレイヤーは **cloud-server** レイヤーに追加する必要はありません。

web-clustering
このレイヤーは、埋め込み Infinispan ベースの Web セッションクラスタリングをプロビジョニングされたサーバーに追加します。

8.2. JBOSS EAP でのユーザーによるレイヤーのプロビジョニング

Red Hat から利用可能なレイヤーのプロビジョニングを行うに加え、開発するカスタムレイヤーをプロビジョニングできます。

手順

1. Galleon Maven プラグインを使用してカスタムレイヤーを構築します。
   詳細は、Maven プロジェクトの準備 を参照してください。
2. アクセス可能な Maven リポジトリーにカスタムレイヤーをデプロイします。
3. カスタム Galleon 機能パック環境変数を使用して、S2I イメージビルドプロセス中に Galleon 機能パックとレイヤーをカスタマイズできます。
   Galleon 機能パックとレイヤーのカスタマイズの詳細は、S2I ビルド時のカスタム Galleon 機能パックの使用 を参照してください。
4. **オプション**: ユーザー定義のレイヤーとサポートされる JBoss EAP レイヤーを参照するカスタムプロビジョニングファイルを作成し、これをアプリケーションディレクトリーに保存します。
   カスタムプロビジョニングファイルの作成の詳細は、Galleon プロビジョニングファイル を参照してください。
5. S2I プロセスを実行して、OpenShift で JBoss EAP サーバーをプロビジョニングします。
   詳細は、S2I ビルド時のカスタム Galleon 機能パックの使用 を参照してください。

8.2.1. JBoss EAP のカスタム Galleon レイヤーのビルドと使用

カスタム Galleon レイヤーは、JBoss EAP 8.0 で動作するように設計された Galleon 機能パック内にパッケージ化します。

Openshift では、JBoss EAP 8.0 サーバー用の MariaDB ドライバーやデータソースなどをプロビジョニングするためのレイヤーを含む Galleon 機能パックをビルドして使用できます。レイヤーには、サーバーにインストールされているコンテンツが含まれています。レイヤーは、サーバーの XML 設定ファイルを更新し、コンテンツをサーバーのインストールに追加できます。

このセクションでは、OpenShift で JBoss EAP 8.0 サーバー用の MariaDB ドライバーとデータソースをプロビジョニングするためのレイヤーを含む Galleon 機能パックをビルドおよび使用する方法について説明します。

8.2.1.1. Maven プロジェクトの準備

Galleon 機能パックは、Maven を使用して作成されます。この手順には、新しい Maven プロジェクトを作成する手順が含まれています。

手順

1. 次のコマンドを実行して、新しい Maven プロジェクトを作成します。

```
mvn archetype:generate -DarchetypeGroupId=org.codehaus.mojo.archetypes -
DarchetypeArtifactId=pom-root -DgroupId=org.jboss.eap.demo -DartifactId=mariadb-galleon-
pack -DinteractiveMode=false
```

2. **mariadb-galleon-pack** ディレクトリーに移動し、**pom.xml** ファイルを更新して Red Hat Maven リポジトリーを含めます。

```
<repositories>
  <repository>
    <id>redhat-ga</id>
    <name>Redhat GA</name>
    <url>https://maven.repository.redhat.com/ga/</url>
  </repository>
</repositories>
```

3. **pom.xml** ファイルを更新して、JBoss EAP Galleon 機能パックと MariaDB ドライバーへの依存関係を追加します。

```
<dependencies>
  <dependency>
    <groupId>org.jboss.eap</groupId>
    <artifactId>wildfly-ee-galleon-pack</artifactId>
    <version>8.0.0.GA-redhat-00010</version>
    <type>zip</type>
  </dependency>
  <dependency>
    <groupId>org.mariadb.jdbc</groupId>
    <artifactId>mariadb-java-client</artifactId>
    <version>2.7.2</version>
  </dependency>
</dependencies>
```

注記

- **<version>A.B.C-redhat-XXXXX</version> A.B.C** はリリース番号、**XXXXX** は JBoss EAP インスタンスのビルド番号です。JBoss EAP リリースのバージョンの詳細は、Red Hat Maven リポジトリーを参照してください。リリース番号とビルド番号は、すべての JBoss EAP リリースで利用できます。https://maven.repository.redhat.com/earlyaccess/all/org/jboss/eap/wildfly-ee-galleon-pack/.

4. **pom.xml** ファイルを更新して、Galeon 機能パックのビルドに使用される Maven プラグインを含めます。

```
<build>
  <plugins>
    <plugin>
      <groupId>org.wildfly.galleon-plugins</groupId>
      <artifactId>wildfly-galleon-maven-plugin</artifactId>
      <version>6.4.8.Final-redhat-00001</version>
```

```
      <executions>
        <execution>
          <id>mariadb-galleon-pack-build</id>
          <goals>
            <goal>build-user-feature-pack</goal>
          </goals>
          <phase>compile</phase>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>
```

8.2.1.2. 機能パックコンテンツの追加

この手順は、カスタム Galleon 機能パック (MariaDB ドライバーとデータソースレイヤーを含む機能パックなど) にレイヤーを追加するのに役立ちます。

前提条件

- Maven プロジェクトを作成している。詳細は、Maven プロジェクトの準備 を参照してください。

手順

1. カスタム機能パック Maven プロジェクト内に **src/main/resources** ディレクトリーを作成します。Maven プロジェクトの準備 を参照してください。このディレクトリーは、機能パックのコンテンツを含むルートディレクトリーです。
2. ディレクトリー **src/main/resources/modules/org/mariadb/jdbc/main** を 作成します。
3. **main** ディレクトリーに、次の内容の **module.xml** という名前のファイルを作成します。

```
<?xml version="1.0" encoding="UTF-8"?>
<module name="org.mariadb.jdbc" xmlns="urn:jboss:module:1.8">
  <resources>
    <artifact name="${org.mariadb.jdbc:mariadb-java-client}"/> 1
  </resources>
  <dependencies> 2
    <module name="java.se"/>
    <module name="jakarta.transaction.api"/>
    <module name="jdk.net"/>
  </dependencies>
</module>
```

1. MariaDB ドライバーの **groupId** と **artifactId**。プロビジョニング時に、実際のドライバー JAR ファイルがインストールされます。ドライバーのバージョンは、**pom.xml** ファイルから参照されます。
2. MariaDB ドライバーの **JBoss Modules** モジュールの依存関係。

4. ディレクトリー **src/main/resources/layers/standalone/**を 作成します。これは、ガレオン機能パックが定義しているすべてのレイヤーのルートディレクトリーです。

5. ディレクトリー **src/main/resources/layers/standalone/mariadb-driver** を 作成します。

6. **mariadb-driver** ディレクトリーで、次の内容で **layer-spec.xml** ファイルを作成します。

```
<?xml version="1.0" ?>
<layer-spec xmlns="urn:jboss:galleon:layer-spec:1.0" name="mariadb-driver">
  <feature spec="subsystem.datasources"> (1)
    <feature spec="subsystem.datasources.jdbc-driver">
      <param name="driver-name" value="mariadb"/>
      <param name="jdbc-driver" value="mariadb"/>
      <param name="driver-xa-datasource-class-name"
value="org.mariadb.jdbc.MariaDbDataSource"/>
      <param name="driver-module-name" value="org.mariadb.jdbc"/>
    </feature>
  </feature>
  <packages> (2)
    <package name="org.mariadb.jdbc"/>
  </packages>
</layer-spec>
```

(1) モジュール **org.mariadb.jdbc** によって実装された **MariaDB** という名前の JDBC ドライバーを使用して、**datasources** サブシステムの設定を更新します。

(2) レイヤーのプロビジョニング時にインストールされるドライバークラスを含む **JBoss Modules** モジュール。

mariadb-driver レイヤーは、**JBoss Modules** モジュールによって実装された JDBC ドライバーの設定で **datasources** サブシステムを更新します。

7. ディレクトリー **src/main/resources/layers/standalone/mariadb-datasource** を作成します。

8. **mariadb-datasource** ディレクトリーで、次の内容で **layer-spec.xml** ファイルを作成します。

```
<?xml version="1.0" ?>
<layer-spec xmlns="urn:jboss:galleon:layer-spec:1.0" name="mariadb-datasource">
  <dependencies>
    <layer name="mariadb-driver"/> (1)
  </dependencies>
  <feature spec="subsystem.datasources.data-source"> (2)
    <param name="data-source" value="MariaDBDS"/>
    <param name="jndi-name"
value="java:jboss/datasources/${env.MARIADB_DATASOURCE:MariaDBDS}"/>
    <param name="connection-url"
value="jdbc:mariadb://${env.MARIADB_HOST:localhost}:${env.MARIADB_PORT:3306}/${env.
MARIADB_DATABASE}"/> (3)
    <param name="driver-name" value="mariadb"/>
    <param name="user-name" value="${env.MARIADB_USER}"/> (4)
    <param name="password" value="${env.MARIADB_PASSWORD}"/>
  </feature>
</layer-spec>
```

(1) この依存関係により、データソースのプロビジョニング時に MariaDB ドライバーのプロビジョニングが強制されます。レイヤーが依存するすべてのレイヤーは、そのレイヤーがプロビジョニングされるときに自動的にプロビジョニングされます。

2 MariaDBDS という名前のデータソースで **datasources** サブシステム設定を更新します。

3 データソースの名前、ホスト、ポート、およびデータベースの値は、サーバーの起動時に設定される環境変数 **MARIADB_DATASOURCE**、**MARIADB_HOST**、**MARIADB_PORT**、および **MARIADB_DATABASE** から解決されます。

4 ユーザー名とパスワードの値は、環境変数 **MARIADB_USER** および **MARIADB_PASSWORD** から解決されます。

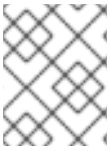
9. 次のコマンドを実行して、Galeon 機能パックをビルドします。

```
mvn clean install
```

ファイル **target/mariadb-galleon-pack-1.0-SNAPSHOT.zip** が作成されます。

8.2.1.3. S2I ビルド時のカスタム Galleon 機能パックの使用

カスタム機能パックは、OpenShift S2I ビルド中に発生する Maven ビルドで使用できるようにする必要があります。これは通常、アクセス可能な Maven リポジトリーに **org.jboss.eap.demo:mariadb-galleon-pack:1.0-SNAPSHOT** などのカスタム機能パックをアーティファクトとしてデプロイすることによって実現されます。



注記

カスタム Galleon 機能パックを使用するための JBoss EAP S2I イメージの設定の詳細は、高度な環境変数を使用した Galleon の設定 を参照してください。

前提条件

- **oc** コマンドラインがインストールされている
- OpenShift クラスターにログインしている
- **Red Hat Container** レジストリーへのアクセスを設定しました。詳細は、Red Hat Container Registry を参照してください。
- カスタムガレオン機能パックを作成しました。詳細は、Maven プロジェクトの準備 を参照してください。

手順

1. 次のコマンドを実行して、**MariaDB** データベースを開始します。この例では、**MariaDB image mariadb-105-rhel7** を使用します。サポートされている最新バージョンの **MariaDB** イメージを使用する必要があります。**MariaDB images** の詳細は、Red Hat Ecosystem Catalog を参照してください。

```
oc new-app -e MYSQL_USER=admin -e MYSQL_PASSWORD=admin -e
MYSQL_DATABASE=mariadb registry.redhat.io/rhscl/mariadb-105-rhel7
```

OpenShift サービス **mariadb-101-rhel7** が作成され、開始されます。

2. Maven プロジェクトディレクトリー **mariadb-galleon-pack** 内で次のコマンドを実行して、カスタム機能パック Maven ビルドによって生成された機能パックアーカイブからシークレットを作成します。

```
oc create secret generic mariadb-galleon-pack --from-file=target/mariadb-galleon-pack-1.0-SNAPSHOT.zip
```

シークレット **mariadb-galleon-pack** が作成されます。S2I ビルドを開始するときに、このシークレットを使用して機能パックの .zip ファイルを Pod にマウントし、サーバーのプロビジョニングフェーズでファイルを使用できるようにします。

8.2.1.4. JBoss EAP 8 イメージストリームのインポート

以下の手順に従って、JBoss EAP 8.0 イメージストリームをインポートできます。

手順

1. JBoss EAP 8.0 イメージストリームをインポートします。

```
oc import-image jboss-eap-8/eap8-openjdk17-builder-openshift-rhel8:latest --from=registry.redhat.io/jboss-eap-8/eap8-openjdk17-builder-openshift-rhel8:latest --confirm
```

8.2.1.4.1. JBoss EAP maven プラグインを使用した S2I ビルドの作成

eap-maven-plugin は、JBoss EAP **galleon feature-pack**、JBoss EAP **cloud galleon feature-pack**、および **mariadb galleon feature-pack** への参照の両方で設定されています。**pom.xml** の抜粋を参照してください。

```
<feature-packs>
  <feature-pack>
    <location>org.jboss.eap:wildfly-ee-galleon-pack</location>
  </feature-pack>
  <feature-pack>
    <location>org.jboss.eap.cloud:eap-cloud-galleon-pack</location>
  </feature-pack>
  <feature-pack>
    <location>org.jboss.eap.demo:mariadb-galleon-pack:1.0-SNAPSHOT</location> (1)
  </feature-pack>
</feature-packs>
<layers>
  <layer>jaxrs-server</layer>
  <layer>mariadb-datasource</layer> (2)
</layers>
```

(1) **mariadb feature-pack** バージョンが必要です。JBoss EAP 8 で設定されたチャネルでは解決されません。

(2) **mariadb-datasource** レイヤー。

手順

1. 次のコマンドを実行して、S2I ビルドを作成します。

```
oc new-build eap8-openjdk17-builder-openshift-rhel8:latest~https://github.com/jboss-container-images/jboss-eap-8-openshift-image#EAP_8.0.0 \
--context-dir=examples/eap/custom-layers/application \
--build-secret=mariadb-galleon-pack:/tmp/demo-maven-repository/org/jboss/eap/demo/mariadb-galleon-pack/1.0-SNAPSHOT \ 1
--name=mariadb-app-build
```

1. **mariadb-galleon-pack** シークレットは **/tmp/demo-maven-repository/org/jboss/eap/demo/mariadb-galleon-pack/1.0-SNAPSHOT** ディレクトリーにマウントされます。

関連情報

詳細は、JBoss EAP 8.0 のデモの例 を参照してください。

8.2.1.4.2. 従来の S2I プロビジョニング機能を使用して S2I ビルドを作成する

サーバーをプロビジョニングできるように、**openshift-legacy** プロファイルを使用して S2I ビルドを設定できます。

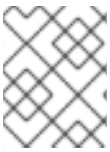
手順

1. 次のコマンドを実行して、新しい OpenShift ビルドを作成します。

```
oc new-build eap8-openjdk17-builder-openshift-rhel8:latest~https://github.com/jboss-container-images/jboss-eap-8-openshift-image#EAP_8.0.0 \
--context-dir=examples/eap/custom-layers/application \
--env=GALLEON_PROVISION_CHANNELS="org.jboss.eap.channels:eap-8.0" \ 1
--env=GALLEON_PROVISION_FEATURE_PACKS="org.jboss.eap:wildfly-ee-galleon-pack,org.jboss.eap.cloud:eap-cloud-galleon-pack,org.jboss.eap.demo:mariadb-galleon-pack:1.0-SNAPSHOT" \ 2
--env=GALLEON_PROVISION_LAYERS="jaxrs-server,mariadb-datasource" \ 3
--env=GALLEON_CUSTOM_FEATURE_PACKS_MAVEN_REPO="/tmp/demo-maven-repository" \ 4
--env=MAVEN_ARGS="-Popenshift-legacy" \ 5
--build-secret=mariadb-galleon-pack:/tmp/demo-maven-repository/org/jboss/eap/demo/mariadb-galleon-pack/1.0-SNAPSHOT \ 6
--name=mariadb-app-build
```

1. この環境変数は、プロビジョニング中に JBoss EAP 8.0 チャネルを使用します。
2. この環境変数は、JBoss EAP 8.0 の **feature-pack**、**cloud feature-pack**、および **mariadb feature-pack** を参照します。
3. この環境変数は、サーバーのプロビジョニングに使用する Galleon 層のセットを参照します。**jaxrs-server** はベースサーバーレイヤーです 。**mariadb-datasource** は、**mariadb** ドライバーと新しいデータソースをサーバーインストールにもたらすカスタムレイヤーです。
4. これは、**mariadb feature-pack** が含まれているローカルの Maven リポジトリーの場所を指します。
5. この環境変数は **MAVEN_ARGS** を再定義して **openshift-legacy** プロファイルを有効にします。

6 **mariadb-galleon-pack** シークレットは **/tmp/demo-maven-repository/org/jboss/eap/demo/mariadb-galleon-pack/1.0-SNAPSHOT** ディレクトリーにマウントされます。



注記

このディレクトリーパスは、パスマッピングへの Maven リポジトリーアーティファクト座標に準拠しています。

8.2.1.4.3. ビルドの開始

新しいビルドを作成することで、**mariadb-app-build** イメージを作成できます。

手順

1. 以前に作成したものと同じ OpenShift ビルドから新しいビルドを開始し、次のコマンドを実行します。

```
oc start-build mariadb-app-build
```

コマンドの実行が成功すると、イメージ **mariadb-app-build** が作成されます。

8.2.1.4.4. 新しいデプロイメントの作成

データソースを実行中の **MariaDB** データベースにバインドするために必要な環境変数を指定することで、新しいデプロイメントを作成できます。

手順

1. 次のコマンドを実行して、新しいデプロイを作成します。

```
oc new-app --name=mariadb-app mariadb-app-build \
--env=MARIADB_PORT=3306 \
--env=MARIADB_USER=admin \
--env=MARIADB_PASSWORD=admin \
--env=MARIADB_HOST=mariadb-105-rhel7 \
--env=MARIADB_DATABASE=mariadb  \
--env=MARIADB_DATASOURCE=Demo 1
```

1 デモは、データソースの名前が **Demo** であることを想定しています。

注記

カスタム Galleon 機能パック環境変数の詳細は、カスタム Galleon 機能パック環境変数 を参照してください。

2. **mariadb-app** アプリケーションを公開し、次のコマンドを実行します。

```
oc expose svc/mariadb-app
```

3. 新しいタスクを作成するには、次のコマンドを実行します。

```
curl -X POST http://$(oc get route mariadb-app --template='{{ .spec.host }}')/tasks/title/foo
```

4. タスクのリストにアクセスするには、次のコマンドを実行します。

```
curl http://$(oc get route mariadb-app --template='{{ .spec.host }}')
```

追加されたタスクがブラウザーに表示されます。

8.2.2. 高度な環境変数を使用した Galleon の設定

高度なカスタム Galleon 機能パック環境変数を使用して、S2I イメージビルドプロセス中にカスタム Galleon 機能パックとレイヤーを保存する場所をカスタマイズできます。これらの高度なカスタム Galleon 機能パック環境変数は次のとおりです。

- **GALLEON_DIR=<path>**: これは、デフォルトの **<project_root_dir>/galleon** ディレクトリーパスを **<project_root_dir>/<GALLEON_DIR>** に上書きします。
- **GALLEON_CUSTOM_FEATURE_PACKS_MAVEN_REPO = <path>**。これは、**<project root dir>/galleon/repository** ディレクトリーパスを、Maven ローカルリポジトリーキャッシュディレクトリーへの絶対パスでオーバーライドします。このリポジトリーには、カスタム Galleon 機能パックが含まれています。

Galleon 機能パックのアーカイブファイルは、Maven ローカルキャッシュファイルシステム設定に準拠したサブディレクトリー内に配置する必要があります。たとえば、**path-to-repository/org/examples/my-feature-pack/1.0.0.Final/my-feature-pack-1.0.0.Final.zip** パス内の **org.examples:my-feature-pack:1.0.0.Final** 機能パックを見つけます。

<project_root>/<GALLEON_DIR> ディレクトリーに **settings.xml** ファイルを作成することにより、Maven プロジェクト設定を設定できます。**GALLEON_DIR** のデフォルト値は **<project_root_dir>/galleon** です。Maven はこのファイルを使用して、アプリケーション用のカスタム Galleon 機能パックをプロビジョニングします。**settings.xml** ファイルを作成しない場合、Maven は S2I イメージによって作成されたデフォルトの **settings.xml** ファイルを使用します。

重要

S2I ビルダーイメージはローカル Maven リポジトリーの場所を指定するため、**settings.xml** ファイルでローカル Maven リポジトリーの場所を指定しないでください。S2I ビルダーイメージは、S2I ビルドプロセス中にこの場所を使用します。

関連情報

- カスタム Galleon 機能パック環境変数

8.2.3. カスタム Galleon 機能パック環境変数

以下のカスタム Galleon 機能パック環境変数のいずれかを使用して、JBoss EAP S2I イメージの使用方法をカスタマイズできます。

表8.1 カスタム Galleon 機能パック環境変数の説明

環境変数	説明

環境変数	説明
GALLEON_DIR=<path>	ここで、**<path>** は、アプリケーションプロジェクトの root ディレクトリーに相対的なディレクトリーです。**<path>** ディレクトリーには、**settings.xml** ファイルやローカル Maven リポジトリーキャッシュなどのオプションの Galleon カスタムコンテンツが含まれています。このキャッシュには、カスタム Galleon 機能パックが含まれています。 デフォルトのディレクトリーは **galleon** です。
GALLEON_CUSTOM_FEATURE_PACKS_MAVEN_REPO=<path>	**<path>** は、カスタム機能パックを含む Maven ローカルリポジトリーディレクトリーへの絶対パスです。ディレクトリーのデフォルトは **galleon/repository** に設定されます。
GALLEON_PROVISION_FEATURE_PACKS=<list_of_galleon_feature_packs>	**<list_of_galleon_feature_packs>** は、Maven コーディネートによって識別されるカスタム Galleon 機能パックのコンマ区切りリストです。リストする機能パックは、ビルダーイメージに存在する JBoss EAP 8.0 サーバーのバージョンと互換性がある必要があります。 **GALLEON_PROVISION_LAYERS** 環境変数を使用して、カスタム機能パックによって定義された Galleon レイヤーをサーバーに設定できます。

第9章 JBOSS EAP アプリケーションの OPENSHIFT CONTAINER PLATFORM へのデプロイ

9.1. JBOSS EAP OPERATOR による OPENSHIFT へのアプリケーションのデプロイの自動化

EAP Operator は OpenShift API を拡張する JBoss EAP 専用のコントローラーです。EAP Operator を使用することで、複雑なステートフルアプリケーションのインスタンスの作成、設定、管理、およびシームレスなアップグレードを行うことができます。

EAP Operator は、複数の JBoss EAP Java アプリケーションインスタンスをクラスター全体で管理します。また、レプリカを縮小し、削除するために **clean** として Pod をマークする前に、すべてのトランザクションが完了していることを検証することで、アプリケーションクラスターでの安全なトランザクションリカバリーを行えるようにします。EAP Operator は、Jakarta Enterprise Beans リモーティングおよびトランザクションリカバリープロセッシングの適切な処理に **StatefulSet** を使用します。**StatefulSet** は Pod の再起動後もストレージおよびネットワークのホスト名の安定性を永続的に保持します。

EAP Operator をインストールするには、OperatorHub を使用する必要があります。OperatorHub を使用すると、OpenShift クラスター管理者は Operator の検索、インストール、アップグレードを実行できます。

OpenShift Container Platform 4 では、Operator Lifecycle Manager (OLM) を使用して、すべての Operator および複数のクラスターで実行される関連サービスのインストール、更新、管理を行うことができます。

OLM は OpenShift Container Platform 4 で、デフォルトで実行されます。これは、クラスター管理者がクラスターで実行しているオペレーターへのインストール、アップグレード、およびアクセス付与に役立ちます。OpenShift Container Platform Web コンソールでは、クラスター管理者がオペレーターをインストールし、特定のプロジェクトアクセスを付与して、クラスターで利用可能なオペレーターのカタログを使用するための管理画面を利用できます。

オペレーターおよび OLM の詳細は、OpenShift ドキュメント を参照してください。

9.1.1. Web コンソールを使用した EAP Operator のインストール

JBoss EAP クラスター管理者は、OpenShift Container Platform Web コンソールを使用して Red Hat OperatorHub から EAP Operator をインストールできます。その後、EAP Operator を複数の namespace にサブスクライブすることで、クラスター上で開発者が利用できるようにすることができます。

以下では、Web コンソールを使用して EAP Operator をインストールする前に注意する必要がある点をいくつか紹介します。

- インストールモード: **クラスター上のすべての名前空間** を選択し、すべての名前空間にオペレーターをインストールするか、可能であれば個別の名前空間を選択して、選択した名前空間にのみオペレーターをインストールします。
- チャネルの更新: EAP Operator が複数のチャネルで利用可能な場合は、サブスクライブするチャネルを選択できます。たとえば、**stable** チャネル (存在する場合) からデプロイするには、これをリストから選択します。
- 承認ストラテジー: **自動**の更新または**手動**の更新を選択できます。EAP Operator の自動更新を選択した場合は、Operator の新規バージョンが利用可能になると、Operator Lifecycle

Manager (OLM) により EAP Operator の実行中のインスタンスが自動的にアップグレードされます。手動による更新を選択する場合は、オペレーターの新しいバージョンが利用可能になると、OLM は更新要求を作成します。次に、オペレーターが新規バージョンに更新されるように更新要求を手動で承認する必要があります。

注記

以下の手順では、OpenShift Container Platform Web コンソールの変更に応じて変更される可能性があります。最新の手順や最も正確な手順は、**Working with Operators in OpenShift Container Platform** ガイドの Installing from the OperatorHub using the web console を参照してください。

前提条件

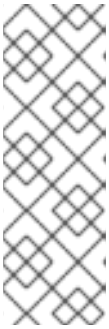
- **cluster-admin** 権限を持つアカウントを使用して OpenShift Container Platform クラスターにアクセスできる。

手順

1. OpenShift Container Platform Web コンソールで、**Operators→ OperatorHub** の順に移動します。
2. 下にスクロールするか、**Filter by keyword** ボックスに **EAP** と入力して EAP Operator を見つけます。
3. JBoss EAP Operator を選択し、**Install** をクリックします。
4. **Create Operator Subscription** ページで以下を行います。
   a. 以下のいずれかを選択します。
      - **クラスター上のすべての名前空間 (デフォルト)** では、デフォルトの **openshift-operators** 名前空間にオペレーターがインストールされ、クラスターのすべての名前空間を監視し、利用できるようにします。このオプションは常に利用できるわけではありません。
      - **クラスター上の特定のネームスペース** では、選択した特定の単一の名前空間にオペレーターがインストールされます。このオペレーターは、この単一名前空間でのみ使用可能となります。

   b. **Update Channel** を選択します。
   c. 前述のように、**Automatic** または **Manual** 承認ストラテジーを選択します。
5. **Subscribe** をクリックし、この OpenShift Container Platform クラスターで選択した名前空間で EAP Operator を利用できるようにします。
   a. 手動での承認ストラテジーを選択した場合は、そのインストールプランを確認して承認するまで、サブスクリプションのアップグレードステータスは **Upgrading** に留まります。**Install Plan** ページでインストールプランを承認すると、サブスクリプションのアップグレードステータスは **Up to date** に変わります。
   b. 自動承認ストラテジーを選択した場合は、アップグレードステータスは介入なしで **Up to date** に変わります。

6. サブスクリプションのアップグレードステータスが **Up to date** になった後に、**Operators → Installed Operators** の順に選択します。そして、EAP ClusterServiceVersion (CSV) が表示され、関連する名前空間で **InstallSucceeded** に **Status** が変わっていることを確認します。



注記

All namespace... インストールモードでは、表示されるステータスは **openshift-operators** 名前空間で **InstallSucceeded** になります。その他の名前空間では、ステータスは **Copied** と表示されます。**Status** フィールドが **InstallSucceeded** に変更されない場合は、さらにトラブルシューティングを行うために問題のレポートを作成する **Workloads → Pods** ページの **openshift-operators** プロジェクト (**A specific namespace...** インストールモードが選択されていた場合は、その他の関連の名前空間) の pod のログを確認してください。

9.1.2. CLI を使用した EAP Operator のインストール

JBoss EAP クラスター管理者は、OpenShift Container Platform CLI を使用して Red Hat OperatorHub から EAP Operator をインストールできます。その後、EAP Operator を複数の namespace にサブスクライブすることで、クラスター上で開発者が利用できるようにすることができます。

CLI を使用して OperatorHub から EAP Operator をインストールする場合は、**oc** コマンドを使用して **Subscription** オブジェクトを作成します。

前提条件

- **cluster-admin** パーミッションを持つアカウントを使用して OpenShift Container Platform クラスターにアクセスできる。
- ローカルシステムに **oc** ツールがインストールされている。

手順

1. OperatorHub からクラスターで利用できる Operator のリストを表示します。

```
$ oc get packagemanifests -n openshift-marketplace | grep eap
NAME        CATALOG               AGE
...
eap         Red Hat Operators     43d
...
```

2. **Subscription** オブジェクト YAML ファイル (例: **eap-operator-sub.yaml**) を作成し、namespace を EAP Operator にサブスクライブします。以下は、**Subscription** オブジェクトの YAML ファイルの例です。

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: eap
  namespace: openshift-operators
spec:
  channel: stable
  installPlanApproval: Automatic
```

```
name: eap 1
source:  redhat-operators 2
sourceNamespace: openshift-marketplace
```

1 サブスクライブするオペレーターの名前。

2 EAP Operator は **redhat-operators** CatalogSource によって提供されます。

チャネルおよび承認ストラテジーの詳細は、この手順の Web コンソール バージョンを参照してください。

3. YAML ファイルから **Subscription** オブジェクトを作成します。

```
$ oc apply -f eap-operator-sub.yaml
$ oc get csv -n openshift-operators
NAME                DISPLAY     VERSION   REPLACES   PHASE
eap-operator.v1.0.0   JBoss EAP   1.0.0               Succeeded
```

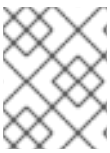
EAP Operator が正常にインストールされます。この時点で、OLM が EAP Operator を認識します。Operator の ClusterServiceVersion (CSV) がターゲット namespace に表示され、EAP Operator が提供する API を作成に使用できるようになります。

9.1.3. EAP Operator を使用した OpenShift への Java アプリケーションのデプロイ

EAP Operator は、OpenShift への Java アプリケーションのデプロイを自動化するのに役立ちます。EAP Operator API の詳細は、EAP Operator: API 情報 を参照してください。

前提条件

- EAP Operator がインストールされている。EAP Operator のインストールに関する詳細は、Web コンソールを使用した EAP Operator のインストール および CLI を使用した EAP Operator のインストール を参照してください。
- JBoss EAP for OpenShift Source-to-Image (S2I) ビルドイメージを使用して、ユーザーアプリケーションの Docker イメージを構築している。
- アプリケーションの CustomResourceDefinition (CRD) ファイルが参照する場合は、**Secret** オブジェクトを作成している。新しい **Secret** オブジェクト作成の詳細は、Secret の作成 を参照してください。
- アプリケーションの CRD ファイルが参照する場合は、**ConfigMap** を作成している。**ConfigMap** 作成の詳細は、ConfigMap の作成 を参照してください。
- 必要であれば、**standalone.xml** ファイルから **ConfigMap** を作成している。**standalone.xml** ファイルから **ConfigMap** を作成する方法の詳細は、standalone.xml ファイルからの ConfigMap の作成 を参照してください。



注記

ConfigMap からの **standalone.xml** ファイルの提供は、JBoss EAP 8.0 ではサポートされていません。

手順

1. Web ブラウザーを開き、OperatorHub にログインします。
2. Java アプリケーションに使用する **Project** または名前空間を選択します。
3. **Installed Operator** に移動し、**JBoss EAP Operator** を選択します。
4. **Overview** タブで、**Create Instance** リンクをクリックします。
5. アプリケーションイメージの詳細を指定します。
   アプリケーションイメージは、Java アプリケーションが含まれる Docker イメージを指定します。イメージは JBoss EAP for OpenShift の Source-to-Image (S2I) ビルドイメージを使用してビルドする必要があります。**applicationImage** フィールドがイメージストリームタグに対応している場合は、イメージへの変更により、アプリケーションの自動アップグレードがトリガーされます。

   JBoss EAP for OpenShift アプリケーションイメージの以下のリファレンスのいずれかを指定できます。

   - イメージの名前: mycomp/myapp
   - タグ: mycomp/myapp:1.0
   - A digest: mycomp/myapp:@sha256:0af38bc38be93116b6a1d86a9c78bd14cd527121970899d719baf78e
   - イメージストリームタグ: my-app:latest
6. アプリケーションのサイズを指定します。以下に例を示します。

```
spec:
  replicas:2
```

7. **env spec** を使用してアプリケーション環境を設定します。環境変数 は、POSTGRESQL_SERVICE_HOST などの値、または POSTGRESQL_USER などの **Secret** オブジェクトから直接取得できます。以下に例を示します。

```
spec:
  env:
  - name: POSTGRESQL_SERVICE_HOST
    value: postgresql
  - name: POSTGRESQL_SERVICE_PORT
    value: '5432'
  - name: POSTGRESQL_DATABASE
    valueFrom:
      secretKeyRef:
        key: database-name
        name: postgresql
  - name: POSTGRESQL_USER
    valueFrom:
      secretKeyRef:
        key: database-user
        name: postgresql
  - name: POSTGRESQL_PASSWORD
    valueFrom:
```

```
    secretKeyRef:
      key: database-password
      name: postgresql
```

8. アプリケーションのデプロイメントに関連する以下のオプションの設定を行います。

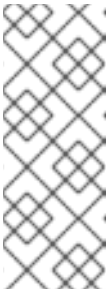
- サーバーデータディレクトリーのストレージ要件を指定します。詳細は、アプリケーションの永続ストレージの設定 を参照してください。

- **WildFlyServerSpec** で作成した **Secret** の名前を指定し、アプリケーションを実行している Pod のボリュームとしてマウントします。以下に例を示します。

```
spec:
  secrets:
    - my-secret
```

Secret は **/etc/secrets/<secret name>** にマウントされ、それぞれのキー/ 値がファイルとして保存されます。ファイルの名前がキーに、コンテンツが値になります。**Secret** は pod 内のボリュームとしてマウントされます。以下の例は、キー値の検索に使用できるコマンドを示しています。

```
$ ls /etc/secrets/my-secret/
my-key  my-password
$ cat /etc/secrets/my-secret/my-key
devuser
$ cat /etc/secrets/my-secret/my-password
my-very-secure-pasword
```



注記

Secret オブジェクトを変更すると、プロジェクトの一貫性が失われることがあります。Red Hat では、既存の **Secret** オブジェクトを変更する代わりに、古いオブジェクトと同じコンテンツを持つ新規オブジェクトを作成することを推奨します。これで、必要に応じてコンテンツを更新し、オペレーターカスタムリソース (CR) の参照を更新できます。これは新しい CR 更新とみなされ、pod はリロードされます。

- **WildFlyServerSpec** で作成した **ConfigMap** の名前を指定し、アプリケーションを実行している Pod のボリュームとしてマウントします。以下に例を示します。

```
spec:
  configMaps:
  - my-config
```

ConfigMap は **/etc/configmaps/<configmap name>** にマウントされ、それぞれのキー/ 値はファイルとして保存されます。ファイルの名前がキーに、コンテンツが値になります。**ConfigMap** は pod 内のボリュームとしてマウントされます。キーの値を検索するには、次のコマンドを実行します。

```
$ ls /etc/configmaps/my-config/
key1 key2
$ cat /etc/configmaps/my-config/key1
```

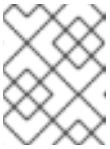
```
value1
$ cat /etc/configmaps/my-config/key2
value2
```

注記

ConfigMap を変更すると、プロジェクトの一貫性が失われることがあります。Red Hat では、既存の **ConfigMap** オブジェクトを変更する代わりに、古いオブジェクトと同じコンテンツを持つ新しい **ConfigMap** を作成することを推奨します。これで、必要に応じてコンテンツを更新し、オペレーターカスタムリソース (CR) の参照を更新できます。これは新しい CR 更新とみなされ、pod はリロードされます。

- 独自のスタンドアロン **ConfigMap** を選択する場合は、 **ConfigMap** の名前と **standalone.xml** ファイルのキーを指定します。

```
standaloneConfigMap:
  name: clusterbench-config-map
  key: standalone.xml
```



注記

standalone.xml ファイルからの **ConfigMap** の作成は、JBoss EAP 8.0 ではサポートされていません。

- OpenShift でデフォルトの HTTP ルートの作成を無効にする場合は、**disableHTTPRoute** を **true** に設定します。

```
spec:
  disableHTTPRoute: true
```

9.1.3.1. シークレットの作成

アプリケーションの CustomResourceDefinition (CRD) ファイルが **Secret** を参照する場合は、EAP Operator を使用してアプリケーションを OpenShift にデプロイする前に **Secret** を作成する必要があります。

手順

- **Secret** を作成するには、以下を実行します。

```
$ oc create secret generic my-secret --from-literal=my-key=devuser --from-literal=my-password='my-very-secure-pasword'
```

9.1.3.2. ConfigMap の作成

アプリケーションの CustomResourceDefinition (CRD) ファイルが **spec.ConfigMaps** フィールドの ConfigMap を参照する場合は、EAP Operator を使用してアプリケーションを OpenShift にデプロイする前に ConfigMap を作成する必要があります。

手順

- configmap を作成するには、以下を実行します。

```
$ oc create configmap my-config --from-literal=key1=value1 --from-literal=key2=value2
configmap/my-config created
```

9.1.3.3. standalone.xml ファイルからの ConfigMap の作成

JBoss EAP for OpenShift Source-to-Image (S2I) から提供されるアプリケーションイメージで使用する代わりに、独自の JBoss EAP スタンドアロン設定を作成できます。**standalone.xml** ファイルは、オペレーターからアクセスできる **ConfigMap** に配置する必要があります。

注記

ConfigMap からの **standalone.xml** ファイルの提供は、JBoss EAP 8.0 ではサポートされていません。

手順

- **standalone.xml** ファイルから **ConfigMap** を作成するには、以下を実行します。

```
$ oc create configmap clusterbench-config-map --from-file
examples/clustering/config/standalone.xml
configmap/clusterbench-config-map created
```

9.1.3.4. アプリケーションの永続ストレージの設定

アプリケーションが一部のデータについて永続ストレージを必要とする場合 (pod の再起動後も維持する必要のあるトランザクションログやメッセージングログなど) は、ストレージ仕様を設定します。ストレージ仕様が空の場合は、**EmptyDir** ボリュームはアプリケーションの各 pod によって使用されます。ただし、このボリュームは、対応する pod が停止した後は使用されなくなります。

手順

1. **volumeClaimTemplate** を指定し、リソース要件を設定して、JBoss EAP スタンドアロンデータディレクトリーを保存します。テンプレートの名前は JBoss EAP の名前から派生します。対応するボリュームは **ReadWriteOnce** アクセスモードでマウントされます。

```
spec:
  storage:
    volumeClaimTemplate:
      spec:
        resources:
          requests:
            storage: 3Gi
```

このストレージ要件を満たす永続ボリュームは、**/eap/standalone/data** ディレクトリーにマウントされます。

9.1.4. EAP Operator を使用したアプリケーションのメトリクスの表示

EAP Operator を使用すると、OpenShift にデプロイされたアプリケーションのメトリクスを表示できます。

クラスター管理者がプロジェクトでメトリクスの監視を有効にすると、EAP Operator によって OpenShift コンソールにメトリクスが自動的に表示されます。

前提条件

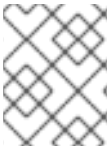
- クラスター管理者が、プロジェクトのモニタリングを有効にしている。詳細は、ユーザー定義プロジェクトのモニタリングの有効化 を参照する。

手順

1. OpenShift Container Platform Web コンソールで、**Monitoring→ Metrics** の順に移動します。
2. **Metrics** 画面で、テキストボックスにアプリケーションの名前を入力してアプリケーションを選択します。アプリケーションのメトリクスが画面に表示されます。

9.1.5. Web コンソールを使用した EAP Operator のアンインストール

EAP Operator はクラスターから削除またはアンインストールできます。サブスクリプションを削除して、サブスクライブされた namespace から EAP Operator を削除できます。EAP Operator の ClusterServiceVersion (CSV) およびデプロイメントを削除することもできます。



注記

データの一貫性と安全性を確保するために、クラスター内の Pod の数を 0 にスケールダウンしてから EAP Operator をアンインストールしてください。

EAP Operator は、Web コンソールを使用してアンインストールできます。

警告

wildflyserver 定義の全体を削除する場合 (**oc delete wildflyserver <deployment_name>**)、トランザクション回復プロセスは開始されず、未完了のトランザクションに関係なく Pod は終了します。この操作により、後で開始するデータの変更がブロックされる可能性があります。この **wildflyserver** を使用したトランザクションエンタープライズ bean リモート呼び出しに関連する他の JBoss EAP インスタンスのデータ変更もブロックされる可能性があります。

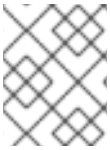
手順

1. **Operators→ Installed Operators** ページから、**JBoss EAP** を選択します。
2. **Operator Details** ページの右側で、**Actions** ドロップダウンメニューから **Uninstall Operator** を選択します。
3. 削除対象のインストールに関連したすべてのコンポーネントが必要であれば、**Remove Operator Subscription** ウインドウでダイアログが表示されたら、**Also completely remove the Operator from the selected namespace** チェックボックスを必要に応じて選択します。これにより CSV が削除され、オペレーターに関連付けられた pod、デプロイメント、カスタムリソース定義 (CRD) およびカスタムリソース (CR) が削除されます。

4. **Remove** をクリックします。EAP Operator が実行を停止し、更新を受信しなくなります。

9.1.6. CLI を使用した JBoss EAP Operator のアンインストール

EAP Operator はクラスターから削除またはアンインストールできます。サブスクリプションを削除して、サブスクライブされた namespace から EAP Operator を削除できます。EAP Operator の ClusterServiceVersion (CSV) およびデプロイメントを削除することもできます。



注記

データの一貫性と安全性を確保するために、クラスター内の Pod の数を 0 にスケールダウンしてから EAP Operator をアンインストールしてください。

EAP Operator は、コマンドラインを使用してアンインストールできます。

コマンドラインを使用する場合は、サブスクリプションと CSV をターゲットの名前空間から削除することにより、オペレーターをアンインストールします。

警告

wildflyserver 定義の全体を削除する場合 (**oc delete wildflyserver <deployment_name>**)、トランザクション回復プロセスは開始されず、未完了のトランザクションに関係なく Pod は終了します。この操作により、後で開始するデータの変更がブロックされる可能性があります。この **wildflyserver** を使用したトランザクションエンタープライズ bean リモート呼び出しに関連する他の JBoss EAP インスタンスのデータ変更もブロックされる可能性があります。

手順

1. **currentCSV** フィールドで EAP Operator サブスクリプションの現在のバージョンを確認します。

```
$ oc get subscription eap-operator -n openshift-operators -o yaml | grep currentCSV
  currentCSV: eap-operator.v1.0.0
```

2. EAP Operator のサブスクリプションを削除します。

```
$ oc delete subscription eap-operator -n openshift-operators
subscription.operators.coreos.com "eap-operator" deleted
```

3. 前のステップの **currentCSV** 値を使用して、ターゲット namespace の EAP Operator の CSV を削除します。

```
$ oc delete clusterserviceversion eap-operator.v1.0.0 -n openshift-operators
clusterserviceversion.operators.coreos.com "eap-operator.v1.0.0" deleted
```

9.1.7. JBoss EAP Operator による安全なトランザクションリカバリー

JBoss EAP Operator は、アプリケーションクラスターを終了する前にデータの整合性を確保します。これを行うために、オペレーターは、レプリカをスケールダウンし、終了のために Pod を **clean** としてマークする前に、すべてのトランザクションが完了していることを確認します。

つまり、データの不整合を起こさずにデプロイメントを安全に削除するには、まず Pod の数を 0 にスケールダウンし、すべての Pod が終了するまで待ってから、**wildflyserver** インスタンスを削除する必要があります。

警告

wildflyserver 定義の全体を削除する場合 (**oc delete wildflyserver <deployment_name>**)、トランザクション回復プロセスは開始されず、未完了のトランザクションに関係なく Pod は終了します。この操作により、後で開始するデータの変更がブロックされる可能性があります。この **wildflyserver** を使用したトランザクションエンタープライズ bean リモート呼び出しに関連する他の JBoss EAP インスタンスのデータ変更もブロックされる可能性があります。

縮小プロセスが開始しても、Pod の状態 (**oc get pod <pod_name>**) は依然として **Running** とマークされています。これは、ターゲット対象のリモートエンタープライズ bean 呼び出しを含む、すべての未完了トランザクションを完了する必要があるためです。

縮小プロセスの状態を監視する場合は、**wildflyserver** インスタンスのステータスを確認します。詳細は、スケールダウンプロセスの監視 を参照してください。スケールダウン中の Pod ステータスの詳細は、スケールダウン中の Pod ステータス を参照してください。

9.1.7.1. 安定したネットワークホスト名の StatefulSets

wildflyserver を管理する EAP Operator は、JBoss EAP Pod を管理する基礎となるオブジェクトとして **StatefulSet** を作成します。

StatefulSet は、ステートフルなアプリケーションを管理するワークロード API オブジェクトです。これは一連の Pod のデプロイメントおよびスケーリングを管理し、これらの Pod の順序と一意性を保証します。

StatefulSet は、クラスターの Pod が事前に定義された順序で命名されるようにします。また、これは Pod が同じ順序で終了することを保証します。たとえば、pod-1 にヒューリスティックな結果のトランザクションがあるとします。つまり、その状態は **SCALING_DOWN_RECOVERY_DIRTY** です。pod-0 が **SCALING_DOWN_CLEAN** の状態であっても、pod-1 の前に終了することはありません。pod-1 が **clean** で、終了するまで、pod-0 は **SCALING_DOWN_CLEAN** 状態に留まります。ただし、pod-0 が **SCALING_DOWN_CLEAN** 状態であっても、新しいリクエストを受け取らず、実際にはアイドル状態になります。

注記

StatefulSet のレプリカサイズを下げたり、Pod 自体を削除したりしても、これらの変更は元に戻りません。

9.1.7.2. スケールダウンプロセスの監視

スケールダウンプロセスの状態を監視する場合は、**wildflyserver** インスタンスのステータスを確認する必要があります。スケールダウン中の各種 Pod ステータスの詳細は、スケールダウン中の Pod ステータス を参照してください。

手順

- 縮小プロセスの状態を確認する方法:

```
oc describe wildflyserver <name>
```

  - **WildFlyServer.Status.Scalingdown Pods** および **WildFlyServer.Status.Replicas** フィールドは、アクティブな Pod とアクティブでない Pod の全体的な状態を示します
  - Scalingdown **Pods** フィールドは、未終了のトランザクションがすべて完了したときに終了する必要のある Pod 数を示します。
  - **WildFlyServer.Status.Replicas** フィールドには、実行中の Pod の現在の数が表示されます。
  - **WildFlyServer.Spec.Replicas** フィールドには、ACTIVE 状態の Pod の数が表示されます。
  - 縮小プロセスに Pod がない場合は、**WildFlyServer.Status.Replicas** と **WildFlyServer.Spec.Replicas** フィールドの Pod の数は同じです。

9.1.7.2.1. スケールダウン中の Pod ステータス

以下の表では、縮小時の各種 Pod ステータスを説明しています。

表9.1 Pod ステータスの説明

Pod ステータス	説明
ACTIVE	Pod はアクティブの状態で、要求を処理しています。
SCALING_DOWN_RECOVERY_INVESTIGATION	Pod はまもなく縮小されます。縮小プロセスが、JBoss EAP のトランザクションの状態について調査中です。
SCALING_DOWN_RECOVERY_DIRTY	JBoss EAP には不完全なトランザクションが含まれています。Pod は、消去されるまで終了しません。トランザクションリカバリープロセスは JBoss EAP で定期的に実行され、トランザクションが完了するまで待機します。
SCALING_DOWN_CLEAN	Pod はトランザクションの縮小処理によって処理され、クラスターからの削除対象として **clean** とマークされます。

9.1.7.3. ヒューリスティックな結果のあるトランザクションの際の縮小

トランザクションの結果が不明な場合は、自動トランザクションリカバリーは行えません。その場合、トランザクションを手動でリカバリーする必要があります。

前提条件

- Pod のステータスが **SCALING_DOWN_RECOVERY_DIRTY** から抜け出せない。

手順

1. CLI を使用して JBoss EAP インスタンスにアクセスします。
2. トランザクションオブジェクトストアのすべてのヒューリスティックトランザクションレコードを解決します。詳細は、**JBoss EAP でのトランザクションの管理** の ヒューリスティックな結果のリカバリー を参照してください。
3. エンタープライズ bean クライアントリカバリーフォルダーからすべてのレコードを削除します。
   a. すべてのファイルを Pod エンタープライズ bean クライアントリカバリーディレクトリーから削除します。

```
$JBOSS_HOME/standalone/data/ejb-xa-recovery
oc exec <podname> rm -rf $JBOSS_HOME/standalone/data/ejb-xa-recovery
```

4. Pod のステータスが **SCALING_DOWN_CLEAN** に変わり、Pod が終了します。

9.1.7.4. トランザクションログに JDBC ストレージを使用するトランザクションサブシステムの設定

システムが **トランザクションログ** を保存するファイルシステムを提供しない場合は、JBoss EAP S2I イメージを使用して JDBC オブジェクトストアを設定します。

重要

JBoss EAP が起動可能な JAR としてデプロイされた場合、S2I 環境変数を使用することはできません。この場合、Galleon レイヤーを作成するか、CLI スクリプトを設定して、必要な設定変更を行う必要があります。

JDBC オブジェクトストアは、環境変数 **TX_DATABASE_PREFIX_MAPPING** で設定できます。この変数の構造は **DB_SERVICE_PREFIX_MAPPING** と同じです。

前提条件

- 環境変数の値に基づいてデータソースを作成している。
- データベースと、JDBC オブジェクトストアで通信する **トランザクションマネージャー** の間で、一貫性のあるデータ読み取りと書き込みパーミッションを確保している。詳細は、configuring JDBC data sources を参照してください。

手順

- S2I 環境変数を使用して JDBC オブジェクトストアをセットアップおよび設定します。

  例:

```
# Narayana JDBC objectstore configuration via s2i env variables
- name: TX_DATABASE_PREFIX_MAPPING
```

```
  value: 'PostgresJdbcObjectStore-postgresql=PG_OBJECTSTORE'
- name: POSTGRESJDBCOBJECTSTORE_POSTGRESQL_SERVICE_HOST
  value: 'postgresql'
- name: POSTGRESJDBCOBJECTSTORE_POSTGRESQL_SERVICE_PORT
  value: '5432'
- name: PG_OBJECTSTORE_JNDI
  value: 'java:jboss/datasources/PostgresJdbc'
- name: PG_OBJECTSTORE_DRIVER
  value: 'postgresql'
- name: PG_OBJECTSTORE_DATABASE
  value: 'sampledb'
- name: PG_OBJECTSTORE_USERNAME
  value: 'admin'
- name: PG_OBJECTSTORE_PASSWORD
  value: 'admin'
```

検証

- **standalone.xml** 設定ファイル **oc rsh <podname> cat /opt/server/standalone/configuration/standalone.xml** を確認することで、データソース設定とトランザクションサブシステム設定の両方を確認できます。
  想定される出力:

```
<datasource jta="false" jndi-name="java:jboss/datasources/PostgresJdbcObjectStore" pool-name="postgresjdbcobjectstore_postgresqlObjectStorePool"
    enabled="true" use-java-context="true" statistics-enabled="${wildfly.datasources.statistics-enabled:${wildfly.statistics-enabled:false}}">
    <connection-url>jdbc:postgresql://postgresql:5432/sampledb</connection-url>
    <driver>postgresql</driver>
    <security>
        <user-name>admin</user-name>
        <password>admin</password>
    </security>
</datasource>

<!-- under subsystem urn:jboss:domain:transactions -->
<jdbc-store datasource-jndi-name="java:jboss/datasources/PostgresJdbcObjectStore">
     <!-- the pod name was named transactions-xa-0 -->
    <action table-prefix="ostransactionsxa0"/>
    <communication table-prefix="ostransactionsxa0"/>
    <state table-prefix="ostransactionsxa0"/>
</jdbc-store>
```

関連情報

- 管理コンソールまたは管理 CLI のいずれかを使用したデータソースの作成に関する詳細は、JBoss EAP**Configuration Guide**の Creating Datasources を参照してください。

9.1.8. Horizontal Pod Autoscaler HPA での Pod の自動スケーリング

EAP Operator を使用すると、水平 Pod オートスケーラー HPA を使用して、その EAP アプリケーションに属する Pod から収集されたメトリクスに基づいて、EAP アプリケーションのスケールを自動的に増減できます。

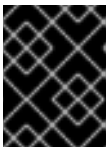
注記

HPA を使用すると、Pod がスケールダウンされた場合でもトランザクションの回復が確実に処理されます。

手順

1. リソースを設定します。

```
apiVersion: wildfly.org/v1alpha1
kind: WildFlyServer
metadata:
  name: eap-helloworld
spec:
  applicationImage: 'eap-helloworld:latest'
  replicas: 1
  resources:
    limits:
      cpu: 500m
      memory: 2Gi
    requests:
      cpu: 100m
      memory: 1Gi
```



重要

自動スケーリングが期待どおりに機能するには、Pod 内のコンテナーのリソース制限とリクエストを指定する必要があります。

2. Horizontal Pod Autoscaler を作成します。

```
oc autoscale wildflyserver/eap-helloworld --cpu-percent=50 --min=1 --max=10
```

検証

- レプリカをチェックすることで、HPA の動作を確認できます。ワークロードの増減に応じて、レプリカの数が増減します。

```
oc get hpa -w
NAME             REFERENCE                       TARGETS    MINPODS   MAXPODS   REPLICAS   AGE
eap-helloworld   WildFlyServer/eap-helloworld   217%/50%   1         10        1          4s
eap-helloworld   WildFlyServer/eap-helloworld   217%/50%   1         10        4          17s
eap-helloworld   WildFlyServer/eap-helloworld   133%/50%   1         10        8          32s
eap-helloworld   WildFlyServer/eap-helloworld   133%/50%   1         10        10         47s
eap-helloworld   WildFlyServer/eap-helloworld   139%/50%   1         10        10         62s
eap-helloworld   WildFlyServer/eap-helloworld   180%/50%   1         10        10         92s
eap-helloworld   WildFlyServer/eap-helloworld   133%/50%   1         10        10         2m2s
```

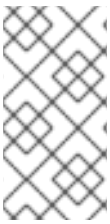
関連情報

- Horizontal Pod Autoscaler での Pod の自動スケーリング

9.1.9. OpenShift 上の Jakarta Enterprise Beans Remoting

9.1.9.1. OpenShift 上の Jakarta Enterprise Beans Remoting

JBoss EAP で、OpenShift 上の各種 JBoss EAP クラスター間でエンタープライズ bean リモーティング呼び出しを正しく機能させるには、OpenShift のエンタープライズ bean リモーティング設定オプションを理解する必要があります。



注記

OpenShift にデプロイする場合は、EAP Operator の使用を検討してください。EAP Operator は、Enterprise Beans リモーティングおよびトランザクションリカバリープロセッシングの適切な処理に **StatefulSet** を使用します。**StatefulSet** は Pod の再起動後もストレージおよびネットワークのホスト名の安定性を永続的に保持します。

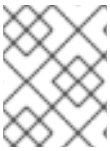
JBoss EAP インスタンスがエンタープライズ bean リモート呼び出しとトランザクション伝搬を使用して通信する場合は、ネットワークホスト名が安定している必要があります。Pod が再起動した場合でも、同じホスト名で JBoss EAP インスタンスに到達できる必要があります。ステートフルなコンポーネントであるトランザクションマネージャーは、永続化されたトランザクションデータを特定の JBoss EAP インスタンスにバインドします。トランザクションログは特定の JBoss EAP インスタンスにバインドされるため、同じインスタンスで完了する必要があります。

JDBC トランザクションログストアが使用されたときにデータの損失を防ぐため、データベースによってデータの一貫性の読み取りおよび書き込みが提供されるようにしてください。一貫性のあるデータの読み取りおよび書き込みは、データベースが複数のインスタンスで水平スケーリングされている場合に重要になります。

エンタープライズ bean リモート呼び出し元では、リモート呼び出しを設定を 2 つの方法で設定できます。

- リモートアウトバウント接続の定義詳細は、リモートアウトバウンド接続の設定 参照してください。
- リモートサーバーで Bean を探すプログラム的な JNDI を使用します。詳細は、Using Remote Jakarta Enterprise Beans Clients を参照してください。

エンタープライズ bean リモート呼び出し設定メソッドに応じて、ターゲットノードのアドレスを表す値を再設定する必要があります。



注記

リモート呼び出しのターゲットエンタープライズ bean の名前は最初の Pod の DNS アドレスでなければなりません。

StatefulSet の動作は Pod の順序によって異なります。Pod の名前は事前に定義された順序で指定されます。たとえば、アプリケーションを 3 つのレプリカにスケーリングする場合、Pod の名前は **eap-server-0**、**eap-server-1**、**eap-server-2** になります。

EAP Operator は、特定の DNS ホスト名が Pod に割り当てられるように ヘッドレスサービス も使用します。アプリケーションが EAP Operator を使用する場合、ヘッドレスサービスは **eap-server-headless** などの名前で作成されます。この場合、最初の Pod の DNS 名は **eap-server-0.eap-server-headless** になります。

ホスト名 **eap-server-0.eap-server-headless** を使用すると、エンタープライズ bean 呼び出しが、クラスターに接続されている EAP インスタンスに到達できるようになります。ブートストラップ接続は Jakarta Enterprise Beans クライアントを初期化するために使用されます。これは、EAP クラスターの構造を次の手順として収集します。

9.1.9.1.1. OpenShift での Jakarta Enterprise Beans の設定

エンタープライズ bean リモーティングの呼び出し元として動作する JBoss EAP サーバーを設定する必要があります。ターゲットサーバーは、エンタープライズ bean リモート呼び出しを受信するパーミッションを持つようにユーザーを設定する必要があります。

前提条件

- OpenShift で JBoss EAP アプリケーションインスタンスをデプロイおよび管理するために、EAP Operator とサポート対象の JBoss EAP for OpenShift S2I イメージを使用している。
- クラスタリングが正しく設定されている。JBoss EAP クラスタリングの詳細は、クラスタリング セクションを参照してください。

手順

1. エンタープライズ bean リモート呼び出しを受信するパーミッションを持つターゲットサーバーにユーザーを作成します。

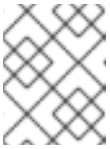
```
$JBOSS_HOME/bin/add-user.sh
```

2. 呼び出し元の JBoss EAP アプリケーションサーバーを設定します。

   a. カスタム設定機能を使用して、**$JBOSS_HOME/standalone/configuration** に **eap-config.xml** ファイルを作成します。詳細は、カスタム設定 を参照してください。

   b. **wildfly.config.url** プロパティーで呼び出し元 JBoss EAP アプリケーションサーバーを設定します。

```
JAVA_OPTS_APPEND="-
Dwildfly.config.url=$JBOSS_HOME/standalone/configuration/eap-config.xml"
```



注記

設定に以下の例を使用する場合は、**>>PASTE_..._HERE<<** を、設定したユーザー名とパスワードで置き換えます。

設定例

```
<configuration>
  <authentication-client xmlns="urn:elytron:1.0">
    <authentication-rules>
      <rule use-configuration="jta">
        <match-abstract-type name="jta" authority="jboss" />
      </rule>
    </authentication-rules>
    <authentication-configurations>
      <configuration name="jta">
        <sasl-mechanism-selector selector="DIGEST-MD5" />
        <providers>
          <use-service-loader />
        </providers>
        <set-user-name name="PASTE_USER_NAME_HERE" />
        <credentials>
```

```
            <clear-password password="PASTE_PASSWORD_HERE" />
          </credentials>
          <set-mechanism-realm name="ApplicationRealm" />
        </configuration>
      </authentication-configurations>
    </authentication-client>
</configuration>
```

第10章 トラブルシューティング

Pod は、さまざまな理由で再起動します。しかし、JBoss EAP Pod の再起動の一般的な原因には OpenShift リソース制約 (特にメモリー不足の問題) が含まれる場合があります。OpenShift Pod エビクション の詳細は、OpenShift ドキュメントを参照してくださ e い。

10.1. POD 再起動のトラブルシューティング

デフォルトでは、JBoss EAP for OpenShift テンプレートは、メモリー不足などの問題が発生すると影響を受けるコンテナーを自動的に再起動するよう設定されています。以下の手順は、メモリー不足やその他の Pod 再起動の問題を診断し、トラブルシューティングを行うのに役立ちます。

1. 問題のある Pod の名前を取得します。
   以下のコマンドを使用すると、Pod の名前と、各 Pod が再起動した回数を確認することができます。

   ```
   $ oc get pods
   ```

2. Pod が再起動した理由を診断するには、前の Pod の JBoss EAP ログまたは OpenShift イベントを調べます。

   a. 前の Pod の JBoss EAP ログを表示するには、以下のコマンドを使用します。

   ```
   oc logs --previous POD_NAME
   ```

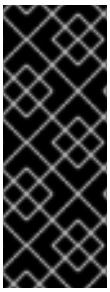
   b. OpenShift イベントを表示するには、以下のコマンドを使用します。

   ```
   $ oc get events
   ```

3. リソースの問題により Pod が再起動される場合は、OpenShift Pod 設定を変更して、その リソース要求および制限 を引き上げることができます。Pod コンピュートリソースの設定 についての詳細は、OpenShift ドキュメントを参照してください。

10.2. JBOSS EAP 管理 CLI を使用したトラブルシューティング

JBoss EAP 管理コンソールである **EAP_HOME/bin/jboss-cli.sh** は、トラブルシューティングの目的でコンテナー内からアクセスすることができます。



重要

JBoss EAP 管理 CLI を使用して、実行中の Pod の設定を変更することは推奨されません。管理 CLI を使用して実行中のコンテナーで変更した設定内容は、コンテナーの再起動時に失われます。

JBoss EAP for OpenShift の設定を変更するには、JBoss EAP サーバーとアプリケーションの設定 を参照してください。

1. 最初に、実行中の Pod にリモートシェルセッションを開きます。

   ```
   $ oc rsh POD_NAME
   ```

2. リモートシェルセッションから以下のコマンドを実行し、JBoss EAP 管理 CLI を起動します。

```
$ /opt/server/bin/jboss-cli.sh
```

10.3. JBOSS EAP 8 で HELM チャートをバージョン 1.0.0 から 1.1.0 に更新するときのエラーのトラブルシューティング

JBoss EAP 8 で Helm チャートを最新バージョンにアップグレードすると、エラーが発生する可能性があります。イミュータブルフィールドを変更してから Helm チャートをアップグレードすると、アップグレード中に次のエラーメッセージが表示される場合があります。

```
UPGRADE FAILED: cannot patch "<helm-release-name>" with kind Deployment: Deployment.apps "
<helm-release-name>" is invalid: spec.selector: Invalid value:
v1.LabelSelector{MatchLabels:map[string]string{"app.kubernetes.io/instance":"<helm-release-
name>", "app.kubernetes.io/name":"<helm-release-name>"}, MatchExpressions:
[]v1.LabelSelectorRequirement(nil)}: field is immutable
```

このエラーを解決するには、コマンド **helm upgrade <helm-release-name>** を実行する前に、コマンド **oc delete deployment <helm-release-name>** を実行してデプロイメントリソースを削除します。

第11章 OPENSHIFT CONTAINER PLATFORM の参照情報

このセクションの内容は、このアプリケーションイメージのエンジニアリングドキュメントから派生したものです。内容は、製品ドキュメントの範囲を超えた開発目的およびテスト用の参考資料として提供されています。

11.1. 情報環境変数

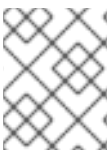
以下の環境変数は、イメージに情報を提供するための環境変数であり、ユーザーが変更すべきではありません。

表11.1 情報環境変数

変数名	説明および値
JBOSS_IMAGE_NAME	イメージ名 値: • **jboss-eap-8/eap8-openjdk17-builder-openshift-rhel8** (JDK 17 / RHEL 8)
JBOSS_IMAGE_VERSION	イメージバージョン。 値: イメージバージョン番号になります。最新の値は Red Hat Container Catalog を参照してください。 • JDK 17 / RHEL 8
JBOSS_MODULES_SYSTEM_PKGS	アプリケーションが利用できる JBoss EAP システムモジュールパッケージのコンマ区切りリスト。 値: **jdk.nashorn.api**
STI_BUILDER	**jee** プロジェクトタイプの OpenShift S2I サポートを提供します。 値: **jee**

11.2. 設定環境変数

以下の環境変数を設定すると再ビルドせずにイメージを調整することができます。



注記

ここに記載されていない他の環境変数については、JBoss EAP のドキュメント を参照してください。

表11.2 設定環境変数

変数名	説明
CLI_GRACEFUL_SHUTDOWN	ゼロでない長さの値が設定された場合、イメージは **TERM** シグナルでシャットダウンしないようにし、JBoss EAP 管理 CLI を使用した **shutdown** コマンドの実行が必要になります。 値の例: **true**
CONTAINER_HEAP_PERCENT	最大 Java ヒープサイズを利用可能なコンテナーメモリーの割合 (パーセント) として設定します。 値の例: **0.5**
CUSTOM_INSTALL_DIRECTORIES	S2I プロセス中にイメージのアーティファクトのインストールおよび設定に使用されるディレクトリーのコンマ区切りリスト。 値の例: **custom,shared**
DEFAULT_JMS_CONNECTION_FACTORY	この値は、Jakarta Messaging 接続ファクトリーのデフォルトの JNDI バインディングを指定するために使用されます (例: **jms-connection-factory='java:jboss/DefaultJMSConnectionFactory'**)。 値の例: **java:jboss/DefaultJMSConnectionFactory**
ENABLE_ACCESS_LOG	標準出力チャネルへのアクセスメッセージのロギングを有効にします。 アクセスメッセージのロギングは以下の方法を使用して実装されます。 ● JBoss EAP 6.4 OpenShift イメージはカスタムの JBoss Web Access Log Valve を使用します。 ● JBoss EAP for OpenShift イメージは、JBoss EAP 7.4 **開発ガイド** の Undertow **AccessLogHandler** を使用します。 デフォルトは **false** です。
INITIAL_HEAP_PERCENT	初期 Java ヒープサイズを最大ヒープサイズの割合 (パーセント) として設定します。 値の例: **0.5**
JAVA_OPTS_APPEND	サーバー起動オプション。 値の例: **-Dfoo=bar**
JBOSS_MODULES_SYSTEM_PKGS_APPEND	**JBOSS_MODULES_SYSTEM_PKGS** 環境変数に追加されるパッケージ名のコンマ区切りリスト。 値の例: **org.jboss.byteman**

変数名	説明
JGROUPS_CLUSTER_PASSWORD	JGroups クラスターへの参加を許可するため、ノードの認証に使用されるパスワード。**ASYM_ENCRYPT** JGroups クラスタートラフィック暗号化プロトコルを使用している場合は**必須**になります。設定がないと、認証は無効になり、クラスターの通信は暗号化されず、警告が発生します。**SYM_ENCRYPT** JGroups クラスタートラフィック暗号化プロトコルを使用する場合は任意です。 値の例: **mypassword**
JGROUPS_ENCRYPT_KEYSTORE	**SYM_ENCRYPT** JGroups クラスタートラフィック暗号化プロトコルを使用する際に指定された、作成済みシークレット内のキーストアファイルの名前。設定しないと、クラスター通信は暗号化されず、警告が発生します。 値の例: **jgroups.jceks**
JGROUPS_ENCRYPT_KEYSTORE_DIR	キーストアを含むシークレットがマウントされているディレクトリーパス。 値の例: **/etc/jgroups-encrypt-secret-volume**
JGROUPS_ENCRYPT_NAME	**SYM_ENCRYPT** JGroups クラスタートラフィック暗号化プロトコルを使用する場合のサーバー証明書に関連する名前。設定しないと、クラスター通信は暗号化されず、警告が発生します。 値の例: **jgroups**
JGROUPS_ENCRYPT_PASSWORD	**SYM_ENCRYPT** JGroups クラスタートラフィック暗号化プロトコルを使用する場合にキーストアおよび証明書へのアクセスに使用されるパスワード。設定しないと、クラスター通信は暗号化されず、警告が発生します。 値の例: **mypassword**
JGROUPS_ENCRYPT_PROTOCOL	クラスタートラフィックの暗号化に使用する JGroups プロトコル。**SYM_ENCRYPT** または **ASYM_ENCRYPT** のいずれかになります。 デフォルトは **SYM_ENCRYPT** です。 値の例: **ASYM_ENCRYPT**
JGROUPS_PING_PROTOCOL	ノードの検索に使用する JGroups プロトコル。**dns.DNS_PING** または **kubernetes.KUBE_PING** のいずれかを使用できます。

変数名	説明
MQ_SIMPLE_DEFAULT_PHYSICAL_DESTINATION	後方互換性を維持するには、**true** を設定し、**queue/MyQueue** および **topic/MyTopic** の代わりに **MyQueue** および **MyTopic** を物理宛先名のデフォルトとして使用します。
OPENSHIFT_DNS_PING_SERVICE_NAME	DNS 検索メカニズムに対してサーバーで ping ポートを公開するサービスの名前。 値の例: **eap-app-ping**
OPENSHIFT_DNS_PING_SERVICE_PORT	DNS 検索メカニズムの ping ポートのポート番号。指定のない場合は、サービスの SRV レコードからポート番号を検出を試み、検出できない場合はデフォルトの **8888** が使用されます。 デフォルトは **8888** です。
OPENSHIFT_KUBE_PING_LABELS	Kubernetes 検索メカニズムのクラスタリングラベルセレクター。 値の例: **app=eap-app**
OPENSHIFT_KUBE_PING_NAMESPACE	Kubernetes 検索メカニズムのクラスタリングプロジェクト namespace。 値の例: **myproject**
SCRIPT_DEBUG	**true** に設定すると、Bash スクリプトが **-x** オプションで実行され、実行と同時にコマンドとその引数が出力されます。

11.3. 公開されたポート

表11.3 公開されたポート

ポート番号	説明
8443	HTTPS

11.4. DATASOURCES

データソースは、環境変数の一部の値を元にして自動的に作成されます。

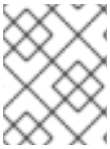
最も重要な環境変数は、データソースの JNDI マッピングを定義する **DB_SERVICE_PREFIX_MAPPING** です。この変数で使用できる値は、**POOLNAME-DATABASETYPE=PREFIX** トリプレットのコンマ区切りリストです。 説明を以下に示します。

- **POOLNAME** はデータソースの **pool-name** として使用されます。
- **DATABASETYPE** は使用するデータベースドライバーです。

- **PREFIX** は、データソースを設定するために使用される環境変数の名前に使用される接頭辞です。

11.4.1. データソースの JNDI マッピング

起動スクリプトは、イメージの起動時に実行される個別のデータソースを **DB_SERVICE_PREFIX_MAPPING** 環境変数に定義された各 **POOLNAME-DATABASETYPE=PREFIX** トリプレットに対して作成します。



注記

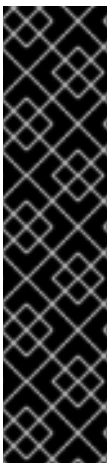
DB_SERVICE_PREFIX_MAPPING の最初の部分 (等号の前) は小文字である必要があります。

DATABASETYPE はデータソースのドライバーを決定します。

ドライバーの設定に関する詳細は、モジュール、ドライバー、および汎用デプロイメント を参照してください。JDK 8 イメージにはデフォルトで設定された **postgresql** および **mysql** のドライバーがあります。

警告

POOLNAME パラメーターには特殊文字を使用しないでください。



データベースドライバー

Red Hat が提供する内部データソースドライバーを JBoss EAP for OpenShift イメージと使用する場合のサポートは、非推奨になりました。Red Hat では、データベースベンダーから取得した JDBC ドライバーを JBoss EAP アプリケーションに使用することを推奨します。

以下の内部データソースは、JBoss EAP for OpenShift イメージでは提供されないようになりました。

- MySQL
- PostgreSQL

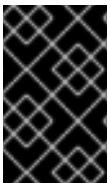
ドライバーのインストールに関する詳細は、モジュール、ドライバー、および汎用デプロイメント を参照してください。

JBoss EAP で JDBC ドライバーを設定するための詳細は、**設定ガイド**の JDBC ドライバー を参照してください。

プロビジョニングされたサーバーに追加する場合は、カスタムレイヤーを作成してこれらのドライバーおよびデータソースをインストールすることもできます。

11.4.1.1. データソース設定環境変数

その他のデータソースプロパティーを設定するには、以下の環境変数を使用します。



重要

必ず **POOLNAME**、**DATABASETYPE**、および **PREFIX** の値を、以下の変数名と適切な値に置き換えてください。置き換え可能な値の説明は、このセクションと データソース セクションに記載されています。

変数名	説明
POOLNAME_DATABASETYPE_SERVICE_HOST	データソースの **connection-url** プロパティーで使用されるデータベースサーバーのホスト名または IP アドレスを定義します。 値の例: **192.168.1.3**
POOLNAME_DATABASETYPE_SERVICE_PORT	データソースのデータベースサーバーのポートを定義します。 値の例: **5432**
PREFIX_BACKGROUND_VALIDATION	**true** に設定すると、データベース接続は使用前にバックグラウンドスレッド周期的に検証されます。デフォルトは **false** で、**validate-on-match** がデフォルトで有効になります。
PREFIX_BACKGROUND_VALIDATION_MILLIS	**background-validation** データベース接続の検証メカニズムが有効である場合 (**PREFIX_BACKGROUND_VALIDATION** 変数が **true** に設定)、検証の頻度をミリ秒単位で指定します。デフォルトは **10000** です。
PREFIX_CONNECTION_CHECKER	使用中の特定のデータベースの接続を検証するために使用される接続チェッカークラスを指定します。 値の例: **org.jboss.jca.adapters.jdbc.extensions.postgres.PostgreSQLValidConnectionChecker**
PREFIX_DATABASE	データソースのデータベース名を定義します。 値の例: **myDatabase**
PREFIX_DRIVER	データソースの Java データベースドライバーを定義します。 例の値: **postgresql**
PREFIX_EXCEPTION_SORTER	致命的なデータベース接続例外の発生後に適切に検出およびクリーンアップを行うために使用される例外ソータークラスを指定します。 例の値: **org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLExceptionSorter**

変数名	説明
PREFIX_JNDI	データソースの JNDI 名を定義します。デフォルトは **java:jboss/datasources/POOLNAME_DATABASETYPE** で、**POOLNAME** および **DATABASETYPE** は上記のトリプレットから取得されます。この設定は、デフォルトの生成された JNDI 名をオーバーライドする場合に便利です。 値の例: **java:jboss/datasources/test-postgresql**
PREFIX_JTA	XA 以外のデータソースの Jakarta Transactions オプションを定義します。XA データソースはデフォルトで機能する Jakarta Transactions です。 デフォルト値は **true** です。
PREFIX_MAX_POOL_SIZE	データソースの最大プールサイズオプションを定義します。 値の例: **20**
PREFIX_MIN_POOL_SIZE	データソースの最小プールサイズオプションを定義します。 値の例: **1**
PREFIX_NONXA	データソースを非 XA データソースとして定義します。デフォルトは **false** です。
PREFIX_PASSWORD	データソースのパスワードを定義します。 値の例: **password**
PREFIX_TX_ISOLATION	データソースの java.sql.Connection トランザクション分離レベルを定義します。 値の例: **TRANSACTION_READ_UNCOMMITTED**
PREFIX_URL	データソースの接続 URL を定義します。 値の例: **jdbc:postgresql://localhost:5432/postgresdb**
PREFIX_USERNAME	データソースのユーザー名を定義します。 値の例: **admin**

11.4.1.2. 例

これらの例は、**DB_SERVICE_PREFIX_MAPPING** 環境変数の値がどのようにデータソースの作成に影響するかを表しています。

11.4.1.2.1. 単一のマッピング

値 **test-postgresql=TEST** について考えてみます。

これは、**java:jboss/datasources/test_postgresql** 名でデータソースを作成します。さらに、パスワードやユーザー名などの必要な設定すべてが、**TEST_USERNAME** や **TEST_PASSWORD** のように、**TEST_** 接頭辞を持つ環境変数として提供されることが想定されます。

11.4.1.2.2. 複数のマッピング

複数のデータソースマッピングを指定できます。

注記

複数のデータソースマッピングは常にコンマで区切ります。

DB_SERVICE_PREFIX_MAPPING 環境変数の値として **cloud-postgresql=CLOUD,test-mysql=TEST_MYSQL** を考慮します。

これは以下の 2 つのデータソースを作成します。

1. **java:jboss/datasources/test_mysql**
2. **java:jboss/datasources/cloud_postgresql**

TEST_MYSQL 接頭辞は、**TEST_MYSQL_USERNAME** のように MySQL データソースのユーザー名やパスワードなどの設定に使用できます。PostgreSQL データソースの場合は、**CLOUD_USERNAME** のように **CLOUD_** 接頭辞を使用します。

11.5. クラスタリング

11.5.1. JGroups 検索メカニズムの設定

OpenShift で JBoss EAP クラスタリングを有効にするには、JBoss EAP 設定の JGroups プロトコルスタックを設定し、**kubernetes.KUBE_PING** または **dns.DNS_PING** 検索メカニズムのいずれかを使用するようにします。

カスタムの **standalone.xml** 設定ファイルを使用することもできますが、環境変数 を使用してイメージビルドで JGroups を設定することが推奨されます。

以下の手順は、環境変数を使用して JBoss EAP for OpenShift イメージの検出メカニズムを設定します。

重要

Helm チャートを使用して JBoss EAP for OpenShift イメージの上にアプリケーションをデプロイする場合、デフォルトの検出メカニズムは **dns.DNS_PING** です。

dns.DNS_PING および **kubernetes.KUBE_PING** 検索メカニズムは互換性がありません。検索に **dns.DNS_PING** メカニズムを使用する 1 つの独立した子クラスターと、**kubernetes.KUBE_PING** メカニズムを使用するもう 1 つの独立した子クラスターを使用して、スーパークラスターを設定することは不可能です。同様に、ローリングアップグレードを実行する場合は、ソースクラスターとターゲットクラスターの両方で同じ検索メカニズムを使用する必要があります。

11.5.1.1. KUBE_PING の設定

KUBE_PING JGroups 検索メカニズムを使用するには、以下を行います。

1. **KUBE_PING** を検索メカニズムとして使用するよう JGroups プロトコルスタックを設定する必要があります。
これには、**JGROUPS_PING_PROTOCOL** 環境変数を **kubernetes.KUBE_PING** に設定します。

```
JGROUPS_PING_PROTOCOL=kubernetes.KUBE_PING
```

2. **KUBERNETES_NAMESPACE** 環境変数を OpenShift プロジェクト名に設定する必要があります。以下に例を示します。

```
KUBERNETES_NAMESPACE=PROJECT_NAME
```

3. **KUBERNETES_LABELS** 環境変数を設定する必要があります。これは サービスレベルで設定したラベル と一致する必要があります。設定がないと、アプリケーション外部の Pod (namespace にあっても) は参加を試みます。以下に例を示します。

```
KUBERNETES_LABELS=application=APP_NAME
```

4. Pod が実行しているサービスアカウントを承認し、Kubernetes の REST API アカウントへのアクセスを許可する必要があります。これは OpenShift CLI を使用して行われます。以下は、現在のプロジェクトの名前空間で **default** サービスアカウントを使用した例になります。

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

プロジェクトの namespace で **eap-service-account** を使用した例は次のとおりです。

```
oc policy add-role-to-user view system:serviceaccount:$(oc project -q):eap-service-account -n $(oc project -q)
```

注記

サービスアカウントへのポリシーの追加の詳細は、アプリケーションのデプロイに向けた OpenShift の準備 を参照してください。

11.5.1.2. DNS_PING の設定

DNS_PING JGroups 検索メカニズムを使用するには、以下を行います。

1. **DNS_PING** を検索メカニズムとして使用するよう JGroups プロトコルスタックを設定する必要があります。
これには、**JGROUPS_PING_PROTOCOL** 環境変数を **dns.DNS_PING** に設定します。

```
JGROUPS_PING_PROTOCOL=dns.DNS_PING
```

2. **OPENSHIFT_DNS_PING_SERVICE_NAME** 環境変数を、クラスターの ping サービスの名前に設定する必要があります。

```
OPENSHIFT_DNS_PING_SERVICE_NAME=PING_SERVICE_NAME
```

3. **OPENSHIFT_DNS_PING_SERVICE_PORT** 環境変数を、ping サービスが公開されるポート番号に設定する必要があります。**DNS_PING** プロトコルは SRV レコードからポートを識別しようとします。 識別できない場合はデフォルトの **8888** になります。

```
OPENSHIFT_DNS_PING_SERVICE_PORT=PING_PORT
```

4. ping ポートを公開する ping サービスを定義する必要があります。このサービスはヘッドレスである必要があり (ClusterIP=None)、以下が必要になります。

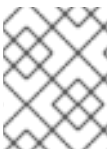
   a. ポートに名前を付ける必要があります。

   b. このサービスは、**service.alpha.kubernetes.io/tolerate-unready-endpoints** および **publishNotReadyAddresses** プロパティー の両方で **true** に設定されたアノテーションを付ける必要があります。

注記

- **service.alpha.kubernetes.io/tolerate-unready-endpoints** プロパティーおよび **publishNotReadyAddresses** プロパティーの両方を使用して、ping サービスが古い OpenShift リリースとそれ以降の OpenShift リリースの両方で機能するようにします。
- これらのアノテーションを省略すると、各ノードは起動時に独自のクラスターを形成します。各ノードは、起動後に他のノードが検出されないため、そのクラスターを起動後に他のノードのクラスターにマージします。

```
kind: Service
apiVersion: v1
spec:
    publishNotReadyAddresses: true
    clusterIP: None
    ports:
    - name: ping
      port: 8888
    selector:
        deploymentConfig: eap-app
metadata:
    name: eap-app-ping
    annotations:
        service.alpha.kubernetes.io/tolerate-unready-endpoints: "true"
        description: "The JGroups ping port for clustering."
```



注記

DNS_PING はサービスアカウントへの変更が必要なく、デフォルトのパーミッションを使用して動作します。

11.5.2. クラスタートラフィックを暗号化するため JGroups を設定

OpenShift で JBoss EAP のクラスタートラフィックを暗号化するには、**SYM_ENCRYPT** または **ASYM_ENCRYPT** プロトコルのいずれかを使用するよう、JBoss EAP 設定の JGroups プロトコルスタックを設定する必要があります。

カスタムの **standalone.xml** 設定ファイルを使用することもできますが、環境変数 を使用してイメージビルドで JGroups を設定することが推奨されます。

以下の手順は、環境変数を使用して、JBoss EAP for OpenShift イメージのクラスタートラフィックの暗号化にプロトコルを設定します。

重要

SYM_ENCRYPT および **ASYM_ENCRYPT** プロトコルは互換性がありません。クラスタートラフィックの暗号化に **SYM_ENCRYPT** を使用する 1 つの独立した子クラスターと、**ASYM_ENCRYPT** プロトコルを使用する別の独立した子クラスターの 2 つを使用してスーパークラスターを設定するのは不可能です。同様に、ローリングアップグレードを実行する場合は、ソースおよびターゲットクラスターの両方で同じプロトコルを使用する必要があります。

11.5.2.1. SYM_ENCRYPT の設定

SYM_ENCRYPT プロトコルを使用して JGroups クラスタートラフィックを暗号化するには、以下を行います。

1. **SYM_ENCRYPT** を暗号化プロトコルをして使用するよう、JGroups プロトコルスタックを設定する必要があります。
   これには、**JGROUPS_ENCRYPT_PROTOCOL** 環境変数を **SYM_ENCRYPT** に設定します。

   ```
   JGROUPS_ENCRYPT_PROTOCOL=SYM_ENCRYPT
   ```

2. **JGROUPS_ENCRYPT_KEYSTORE_DIR** 環境変数は、キーストアを含むシークレットがマウントされているディレクトリーパスに設定する必要があります。以下に例を示します。

   ```
   JGROUPS_ENCRYPT_KEYSTORE_DIR=/etc/jgroups-encrypt-secret-volume
   ```

3. **JGROUPS_ENCRYPT_KEYSTORE** 環境変数は、指定された作成済みシークレット内のキーストアファイルの名前に設定する必要があります。設定しないと、クラスター通信は暗号化されず、警告が発生します。以下に例を示します。

   ```
   JGROUPS_ENCRYPT_KEYSTORE=jgroups.jceks
   ```

4. **JGROUPS_ENCRYPT_NAME** 環境変数を、サーバーの証明書に関連する名前に設定する必要があります。設定しないと、クラスター通信は暗号化されず、警告が発生します。以下に例を示します。

   ```
   JGROUPS_ENCRYPT_NAME=jgroups
   ```

5. **JGROUPS_ENCRYPT_PASSWORD** 環境変数を、キーストアおよび証明書にアクセスするために使用されるパスワードに設定する必要があります。設定しないと、クラスター通信は暗号化されず、警告が発生します。以下に例を示します。

   ```
   JGROUPS_ENCRYPT_PASSWORD=mypassword
   ```

11.5.2.2. ASYM_ENCRYPT の設定

注記

JBoss EAP 8.0 には、新しいバージョンの **ASYM_ENCRYPT** プロトコルが組み込まれています。以前のバージョンのプロトコルは非推奨になりました。**JGROUPS_CLUSTER_PASSWORD** 環境変数を指定する場合は、プロトコルの非推奨バージョンが使用され、警告が Pod ログに出力されます。

ASYM_ENCRYPT プロトコルを使用して JGroups クラストラフィックを暗号化するには、**ASYM_ENCRYPT** を暗号化プロトコルとして指定します。また、**elytron** サブシステムに設定されたキーストアを使用するよう設定します。

```
-e JGROUPS_ENCRYPT_PROTOCOL="ASYM_ENCRYPT" \
-e JGROUPS_ENCRYPT_NAME="encrypt_name" \
-e JGROUPS_ENCRYPT_PASSWORD="encrypt_password" \
-e JGROUPS_ENCRYPT_KEYSTORE="encrypt_keystore" \
-e JGROUPS_CLUSTER_PASSWORD="cluster_password"
```

11.5.3. Pod のスケールアップに関する考慮事項

JGroups の検出メカニズムに基づいて、開始ノードは既存のクラスターコーディネーターノードを検索します。指定されたタイムアウト内にコーディネーターノードが見つからない場合、開始ノードは自分が最初のメンバーであると見なし、コーディネーターステータスを取得します。

複数のノードが同時に起動すると、すべてのノードが最初のメンバーであると見なされ、複数のパーティションを持つ分割クラスターが作成されます。たとえば、**DeploymentConfig** API を使用して Pod を 0 から 2 にスケールアップすると、分割クラスターが作成される場合があります。この状況を回避するには、最初の Pod を開始してから、必要な数の Pod にスケールアップする必要があります。

注記

デフォルトでは、JBoss EAP Operator は **StatefulSet** API を使用します。これにより、Pod の作成が順番に、つまり 1 つずつ開始されるため、分割クラスターの作成が防止されます。

11.6. ネイティブヘルスチェック

JBoss EAP for OpenShift イメージは、OpenShift にデフォルトで含まれている Liveness プローブと Readiness プローブを実装します。詳細は、**OpenShift Container Platform 開発者ガイド** の Liveness および Readiness プローブ を参照してください。

以下の表には、これらのヘルスチェックに合格するために必要な値が記載されています。以下の値以外の場合は、ヘルスチェックに合格せず、イメージの再起動ポリシーにしたがってイメージが再起動されます。

表11.4 Liveness および Readiness チェック

実行されたテスト	Liveness	Readiness
Server Status	すべての状態	Running
Boot Errors	None	None

実行されたテスト	Liveness	Readiness
デプロイメントの状態 [a]	N/A または **failed** エントリーなし	N/A または **failed** エントリーなし
ネイティブヘルスチェック	**UP**	**UP**

[a] Deployment Status デプロイメントが存在しない場合、有効な状態は N/A のみ。

11.7. メッセージング

11.7.1. 外部 Red Hat AMQ ブローカーの設定

環境変数で JBoss EAP for OpenShift イメージを設定し、外部 Red Hat AMQ ブローカーに接続できます。

11.8. セキュリティードメイン

新しいセキュリティードメインを設定するには、ユーザーは **SECDOMAIN_NAME** 環境変数を定義する必要があります。

これにより、環境変数の名前が付けられたセキュリティードメインが作成されます。ドメインをカスタマイズするには、以下の環境変数も定義します。

表11.5 セキュリティードメイン

変数名	説明
SECDOMAIN_NAME	追加のセキュリティードメインを定義します。 値の例: **myDomain**
SECDOMAIN_PASSWORD_STACKING	定義した場合、**password-stacking** モジュールオプションが有効になり、値 **useFirstPass** に設定されます。 値の例: **true**
SECDOMAIN_LOGIN_MODULE	使用されるログインモジュール。 デフォルトは **UsersRoles** です。
SECDOMAIN_USERS_PROPERTIES	ユーザー定義が含まれるプロパティーファイルの名前。 デフォルトは **users.properties** です。
SECDOMAIN_ROLES_PROPERTIES	ロール定義が含まれるプロパティーファイルの名前。 デフォルトは **roles.properties** です。

11.9. HTTPS 環境変数

変数名	説明
HTTPS_NAME	**HTTPS_PASSWORD** および **HTTPS_KEYSTORE** と定義された場合、HTTPS を有効にし、SSL 名を設定します。 **keytool -genkey** コマンドで作成した場合は、キーストアのエイリアス名として指定された値である必要があります。 値の例: **example.com**
HTTPS_PASSWORD	**HTTPS_NAME** および **HTTPS_KEYSTORE** と定義された場合、HTTPS を有効にし、SSL キーパスワードを設定します。 値の例: **passw0rd**
HTTPS_KEYSTORE	**HTTPS_PASSWORD** および **HTTPS_NAME** と定義された場合、HTTPS を有効にし、SSL 証明書キーファイルを **EAP_HOME/standalone/configuration** 以下の相対パスに設定します。 値の例: **ssl.key**

11.10. 管理環境変数

表11.6 管理環境変数

変数名	説明
ADMIN_USERNAME	この変数と **ADMIN_PASSWORD** の両方が定義された場合、JBoss EAP の管理ユーザー名に使用されます。 値の例: **eapadmin**
ADMIN_PASSWORD	指定された **ADMIN_USERNAME** のパスワード。 値の例: **passw0rd**

11.11. S2I

イメージには S2I スクリプトと Maven が含まれます。現在 Maven は、OpenShift 上の JBoss EAP ベースのコンテナー (または関連/ 子孫イメージ) にデプロイされるはずのアプリケーションのビルドツールとしてのみサポートされます。

現在、WAR デプロイメントのみがサポートされます。

11.11.1. カスタム設定

イメージのカスタム設定ファイルを追加することが可能です。**configuration/** ディレクトリーに置かれたすべてのファイルは **EAP_HOME/standalone/configuration/** にコピーされます。たとえば、イメー

ジで使用されるデフォルトの設定をオーバーライドするには、カスタムの **standalone.xml** を **configuration/** ディレクトリーに追加します。このようなデプロイメントの 例を参照 してください。

11.11.1.1. カスタムモジュール

カスタムモジュールを追加することが可能です。**modules/** ディレクトリーからのすべてのファイルは **EAP_HOME/modules/** にコピーされます。このようなデプロイメントの 例を参照 してください。

11.11.2. デプロイメントアーティファクト

デフォルトでは、ソースの **target** ディレクトリーからのアーティファクトがデプロイされます。異なるディレクトリーからデプロイするには、BuildConfig 定義の **ARTIFACT_DIR** 環境変数を設定します。**ARTIFACT_DIR** はコンマ区切りのリストです。例:
ARTIFACT_DIR=app1/target,app2/target,app3/target

11.11.3. アーティファクトリポジトリーミラー

Maven のリポジトリーは、すべてのプロジェクト JAR、ライブラリー JAR、プラグイン、またはその他のプロジェクト固有のアーティファクトなど、さまざまな種類のビルドアーティファクトおよび依存関係を保持します。また、S2I ビルドの実行中にアーティファクトのダウンロード元となる場所も指定します。組織では、中央リポジトリーを使用する他に、ローカルカスタムミラーリポジトリーをデプロイすることが一般的です。

ミラーを使用する利点は次のとおりです。

- 地理的に近く、高速な同期ミラーを使用できる。
- リポジトリーの内容をより良く制御できる。
- パブリックサーバーおよびリポジトリーに依存することなく、異なるチーム (開発者、CI) 全体でアーティファクトを共有できる。
- ビルド時間が改善されました。

多くの場合で、リポジトリーマネージャーはミラーへのローカルキャッシュとして機能できます。リポジトリーマネージャーがすでにデプロイされ、**https://10.0.0.1:8443/repository/internal/** で外部アクセス可能な場合、以下のように **MAVEN_MIRROR_URL** 環境変数をアプリケーションのビルド設定に提供すると S2I ビルドはこのマネージャーを使用することができます。

1. **MAVEN_MIRROR_URL** 変数を適用するビルド設定の名前を特定します。

```
oc get bc -o name
buildconfig/eap
```

2. **eap** のビルド設定を **MAVEN_MIRROR_URL** 環境変数で更新します。

```
oc env bc/eap MAVEN_MIRROR_URL="https://10.0.0.1:8443/repository/internal/"
buildconfig "eap" updated
```

3. 設定を確認します。

```
oc env bc/eap --list
# buildconfigs eap
MAVEN_MIRROR_URL=https://10.0.0.1:8443/repository/internal/
```

4. アプリケーションの新しいビルドをスケジュールします。

注記

アプリケーションのビルド中、Maven 依存関係はデフォルトのパブリックリポジトリーではなく、リポジトリーマネージャーからプルされることを確認できます。またビルドの完了後、ビルド中に取得および使用されたすべての依存関係がミラーに追加されたことが確認できます。

11.11.3.1. セキュアなアーティファクトリポジトリーミラー URL

Maven リポジトリーを介した "中間者攻撃" を防ぐために、JBoss EAP ではアーティファクトリポジトリーのミラー URL にセキュアな URL を使用する必要があります。

URL は、安全な http ("https") とセキュアなポートを指定する必要があります。

デフォルトでは、セキュアでない URL を指定すると、エラーが返されます。この動作は、**-Dinsecure.repositories=WARN** プロパティーを使用して上書きできます。

11.11.4. スクリプト

run

このスクリプトは、**standalone-openshift.xml** 設定で JBoss EAP を設定および開始する **standalone.xml** スクリプトを使用します。

assemble

このスクリプトは Maven を使用してソースをビルドして、パッケージ (WAR) を作成し、それを **EAP_HOME/standalone/deployments** ディレクトリーに移動します。

11.11.5. カスタムスクリプト

JBoss EAP を起動する前に、Pod の起動時に実行するカスタムスクリプトを追加できます。

Pod を起動する際に実行するのに有効なスクリプトを追加できます。これには CLI スクリプトが含まれます。

JBoss EAP をイメージから起動するときにスクリプトを含めるには、以下の 2 つのオプションを利用できます。

- postconfigure.sh として実行される configmap をマウントします。
- 指定したインストールディレクトリーに install.sh スクリプトを追加します。

11.11.5.1. カスタムスクリプトを実行するための configmap のマウント

ランタイム時にカスタムスクリプトを既存のイメージ (つまり、すでにビルドされたイメージ) にマウントする際に configmap をマウントします。

configmap をマウントするには、以下を実行します。

1. postconfigure.sh に追加する内容で configmap を作成します。
   たとえば、プロジェクトの root ディレクトリーに **extensions** というディレクトリーを作成して、スクリプト **postconfigure.sh** と **extensions.cli** を含め、次のコマンドを実行します。

```
$ oc create configmap jboss-cli --from-file=postconfigure.sh=extensions/postconfigure.sh --
from-file=extensions.cli=extensions/extensions.cli
```

2. configmap をデプロイメントコントローラー (dc) 経由で Pod にマウントします。

```
$ oc set volume dc/eap-app --add --name=jboss-cli -m /opt/server/extensions -t configmap --
configmap-name=jboss-cli --default-mode='0755' --overwrite
```

postconfigure.sh の例

```
#!/usr/bin/env bash
set -x
echo "Executing postconfigure.sh"
$JBOSS_HOME/bin/jboss-cli.sh --file=$JBOSS_HOME/extensions/extensions.cli
```

extensions.cli の例

```
embed-server --std-out=echo  --server-config=standalone.xml
:whoami
quit
```

11.11.5.2. install.sh を使用したカスタムスクリプトの実行

ビルド時にイメージの一部としてスクリプトを含める場合は、install.sh を使用しします。

install.sh を使用してカスタムスクリプトを実行するには、以下を実行します。

1. s2i ビルド中に使用されるプロジェクトの git リポジトリーで、**.s2i** というディレクトリーを作成します。
2. **s2i** ディレクトリー内に、以下の内容を含む環境ファイルを追加します。

```
$ cat .s2i/environment
CUSTOM_INSTALL_DIRECTORIES=extensions
```

3. **extensions** というディレクトリーを作成します。
4. **extensions** ディレクトリーで、以下のように内容で postconfigure.sh ファイルを作成します (プレースホルダーコードを環境に適したコードに置き換えます)。

```
$ cat extensions/postconfigure.sh
#!/usr/bin/env bash
echo "Executing patch.cli"
$JBOSS_HOME/bin/jboss-cli.sh --file=$JBOSS_HOME/extensions/some-cli-example.cli
```

5. extensions ディレクトリーで、以下のような内容の install.sh というファイルを作成します (プレースホルダーコードを環境に適したコードに置き換えます)。

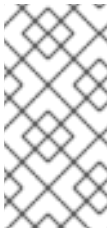
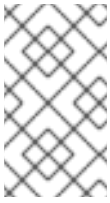
```
$ cat extensions/install.sh
#!/usr/bin/env bash
set -x
echo "Running $PWD/install.sh"
```

```
injected_dir=$1
# copy any needed files into the target build.
cp -rf ${injected_dir} $JBOSS_HOME/extensions
```

11.11.6. 環境変数

s2i build コマンドに指定する環境変数によって、ビルドの実行方法が異なります。指定できる環境変数は次のとおりです。

表11.7 S2I 環境変数

変数名	説明
ARTIFACT_DIR	このディレクトリーからの **.war**、**.ear**、および **.jar** ファイルが **deployments/** ディレクトリーにコピーされます。 値の例: **target**
ENABLE_GENERATE_DEFAULT_DATASOURCE	(オプション)値が **true** の場合、サーバーはデフォルトデータソースでプロビジョニングされます。それ以外の場合は、デフォルトのデータソースは含まれません。
GALLEON_PROVISION_LAYERS	(オプション)S2I プロセスに対し、指定されたレイヤーをプロビジョニングするよう指示します。この値は、プロビジョニングするレイヤーのコンマ区切りのリストです。これには、ベースレイヤーと任意の数のデコレーターレイヤーが含まれます。 値の例: **jaxrs**
GALLEON_PROVISION_CHANNELS	これは JBoss EAP チャネルマニフェストのコンマ区切りリストです。JBoss EAP チャネルマニフェストは、**groupid:artifactId:[version]** によって特定されます。  **注記** バージョンは省略可能です。省略すると、最新のチャネルマニフェストが取得されます。JBoss EAP 8.0 の場合は、**org.jboss.eap.channels:eap-8.0** チャネルが使用されます。
GALLEON_PROVISION_FEATURE_PACKS	S2I イメージのカスタム Galleon 機能パックを指定する環境変数をビルドします。例: **org.jboss.eap:wildfly-ee-galleon-pack:[version]**、**org.jboss.eap.cloud:eap-cloud-galleon-pack:[version]**。  **注記** **GALLEON_PROVISION_CHANNELS=org.jboss.eap.channels:eap-8.0** を設定する場合、機能パックのバージョンは必要ありません。

変数名	説明
HTTP_PROXY_HOST	Maven が使用する HTTP プロキシーのホスト名または IP アドレス。 値の例: **192.168.1.1**
HTTP_PROXY_PORT	Maven が使用する HTTP プロキシーの TCP ポート。 値の例: **8080**
HTTP_PROXY_USERNAME	**HTTP_PROXY_PASSWORD** と指定された場合、HTTP プロキシーのクレデンシャルを使用します。 値の例: **myusername**
HTTP_PROXY_PASSWORD	**HTTP_PROXY_USERNAME** と指定された場合、HTTP プロキシーのクレデンシャルを使用します。 値の例: **mypassword**
HTTP_PROXY_NONPROXYHOSTS	指定された場合、設定された HTTP プロキシーはこれらのホストを無視します。 値の例: **some.example.org|*.example.net**
MAVEN_ARGS	ビルド中に Maven に指定された引数をオーバーライドします。 値の例: **-e -Popenshift -DskipTests -Dcom.redhat.xpaas.repo.redhatga package**
MAVEN_ARGS_APPEND	ビルド中に Maven に指定されたユーザー引数を追加します。 値の例: **-Dfoo=bar**
MAVEN_MIRROR_URL	設定する Maven ミラー/ リポジトリーマネージャーの URL。 値の例: **https://10.0.0.1:8443/repository/internal/** 指定した URL はセキュアである必要があることに注意してください。詳細は、セキュアなアーティファクトリポジトリーミラー URL を参照してください。
MAVEN_CLEAR_REPO	任意で、ビルド後にローカル Maven リポジトリーを消去します。 イメージに存在するサーバーがローカルキャッシュと強く結合されている場合には、キャッシュが削除されず、警告が表示されます。 値の例: **true**

変数名	説明
APP_DATADIR	定義された場合、データファイルのコピー元であるソースのディレクトリー。 値の例: **mydata**
DATA_DIR	**$APP_DATADIR** からのデータがコピーされるイメージのディレクトリー。 値の例: **EAP_HOME/data**

注記

詳細は、OpenShift Container Platform での JBoss EAP アプリケーションのビルドと実行 を参照してください。OpenShift では、JBoss EAP for OpenShift イメージに含まれる Maven と S2I スクリプトが使用されます。

11.12. サポートされないトランザクションリカバリーのシナリオ

- JTS トランザクションは OpenShift ではサポートされていません。
- XTS トランザクションは OpenShift ではサポートされていません。
- 一部のサードパーティーがトランザクションの完了とクラッシュリカバリーフローに使用する XATerminator インターフェイスは、OpenShift ではサポートされていません。
- JBoss Remoting を介して伝播されるトランザクションはサポートされていません。

注記

JBoss Remoting を介して伝播されるトランザクションは、EAP Operator を使用してサポートされます。

11.13. 含まれる JBOSS モジュール

以下の表は、JBoss EAP for OpenShift イメージに含まれる JBoss モジュールを表しています。

表11.8 含まれる JBoss モジュール

JBoss モジュール
org.jboss.as.clustering.common
org.jboss.as.clustering.jgroups
org.jboss.as.ee
org.jgroups

JBoss モジュール
org.openshift.ping
net.oauth.core

11.14. EAP OPERATOR: API 情報

EAP Operator により、以下の API が導入されます。

11.14.1. WildFlyServer

WildFlyServer はカスタム JBoss EAP リソースを定義します。

表11.9 WildFlyServer

フィールド	説明	スキーム	必須
metadata	標準オブジェクトのメタデータ	ObjectMeta v1 meta	false
spec	JBoss EAP デプロイメントの適切な動作の 仕様 。	**WildFlyServerSpec**	true
status	JBoss EAP デプロイメントの最近確認された ステータス 。read-only	WildFlyServerStatus	false

11.14.2. WildFlyServerList

WildFlyServerList は JBoss EAP デプロイメントのリストを定義します。

表11.10 Table

フィールド	説明	スキーム	必須
metadata	標準リストのメタデータ	metav1.ListMeta	false
items	**WildFlyServer** のリスト	**WildFlyServer**	true

11.14.3. WildFlyServerSpec

WildFlyServerSpec は、JBoss EAP リソースの適切な動作の仕様です。

/opt/jboss/wildfly/standalone/data のストレージによって指定されたボリュームをマウントする Pod 仕様で **StatefulSet** を使用します。

表11.11 WildFlyServerSpec

フィールド	説明	スキーム	必須
applicationImage	デプロイされるアプリケーションイメージの名前	string	false
replicas	アプリケーションに必要なレプリカ数	int32]	true
standaloneConfigMap	**ConfigMap** からスタンドアロン設定を読み込む方法を指定する仕様。	**StandaloneConfigMapSpec**	false
resources	ステートフルセットの要求または制限を指定するための **Resources** 仕様。省略した場合、namespace のデフォルトが使用されます。	**Resources**	false
SecurityContext	**SecurityContext** 仕様は、ステートフルセットによって作成された Pod コンテナーの権限とアクセス制御設定を定義します。省略した場合は、デフォルトの権限が使用されます。詳細は、securityContext を参照してください。	***corev1.SecurityContext**	false
storage	ストレージの使用方法を指定するストレージ仕様。省略すると、**EmptyDir** が使用されます (これは Pod の再起動時にデータの永続化されません)。	**StorageSpec**	false
serviceAccountName	JBoss EAP Pod の実行に使用する ServiceAccount の名前	string	false
envFrom	**configMap** または secret のコンテナーに存在する環境変数のリスト	**corev1.EnvFromSource**	false
env	コンテナーに存在する環境変数のリスト	corev1.EnvVar	false

フィールド	説明	スキーム	必須
secrets	コンテナーでボリュームとしてマウントされる secret 名のリスト。各 secret は **/etc/secrets/<secret name>** で読み取り専用ボリュームとしてマウントされます。	string	false
configMaps	コンテナーでボリュームとしてマウントされる **ConfigMap** 名のリスト。各 **ConfigMap** は、**/etc/configmaps/<config map name>** の下に読み取り専用ボリュームとしてマウントされます。	string	false
disableHTTPRoute	アプリケーションサービスの HTTP ポートへのルートの作成を無効にします (省略されている場合は false)。	boolean	false
sessionAffinity	同じクライアント IP からの接続が、毎回同じ JBoss EAP インスタンス/Pod に渡される場合 (省略されている場合は false)	boolean	false

11.14.4. リソース

Resources は、**WildflyServer** リソース用に設定されたリソースを定義します。**Resources** フィールドが定義されていないか、**Request** または **Limits** が空の場合、このリソースは **StatefulSet** から削除されます。このリソースの説明は、標準の **Container** リソースであり、corev1.ResourceRequirements のスキームを使用します。

11.14.5. StorageSpec

StorageSpec は、 **WildFlyServer** リソースに設定されたストレージを定義します。**EmptyDir** も **volumeClaimTemplate** も定義されていない場合は、デフォルトの **EmptyDir** が使用されます。

EAP Operator は、この **StorageSpec** からの情報を使用して **StatefulSet** を設定し、JBoss EAP が独自のデータを永続化するために使用するスタンドアロン/データディレクトリー専用のボリュームをマウントします。たとえば、トランザクションログが考えられます。**EmptyDir** が使用されると、データは

Pod の再起動後も維持されません。JBoss EAP にデプロイされたアプリケーションがトランザクションに依存している場合は、Pod の再起動時に同じ永続ボリュームを再利用できるように **volumeClaimTemplate** を指定します。

表11.12 Table

フィールド	説明	スキーム	必須
emptyDir	JBoss EAP **StatefulSet** によって使用される **EmptyDirVolumeSource**	corev1.EmptyDirVolumeSource	false
volumeClaimTemplate	JBoss EAP スタンドアロンデータディレクトリーを格納するため **Resources** 要件を設定するための PersistentVolumeClaim 仕様。テンプレートの名前は、**WildFlyServer** 名から派生しています。対応するボリュームは **ReadWriteOnce** アクセスモードでマウントされます。	corev1.PersistentVolumeClaim	false

11.14.6. StandaloneConfigMapSpec

StandaloneConfigMapSpec は、JBoss EAP スタンドアロン設定を **ConfigMap** から読み取る方法を定義します。省略すると、JBoss EAP はそのイメージから **standalone.xml** 設定を使用します。

表11.13 StandaloneConfigMapSpec

フィールド	説明	スキーム	必須
name	スタンドアロン設定の XML ファイルを含む **ConfigMap** の名前。	string	true
key	値がスタンドアロン設定の XML ファイルの **ConfigMap** のキー。省略すると、仕様は **standalone.xml** キーを見つけます。	string	false

11.14.7. WildFlyServerStatus

WildFlyServerStatus は、JBoss EAP デプロイメントの最新の確認ステータスです。read-only

表11.14 WildFlyServerStatus

フィールド	説明	スキーム	必須
replicas	アプリケーションの実際のレプリカ数	int32	true
selector	HorizontalPodAutoscaler によって使用される Pod のセレクター	string	true
hosts	アプリケーション HTTP サービスへルーティングするホスト	string	true
pods	Pod のステータス	PodStatus	true
scalingdownPods	スケールダウンのクリーニングプロセス下の Pod 数	int32	true

11.14.8. PodStatus

PodStatus は、JBoss EAP アプリケーションを実行する Pod の最新ステータスです。

表11.15 PodStatus

フィールド	説明	スキーム	必須
name	Pod の名前	string	true
podIP	Pod に割り当てられる IP アドレス	string	true
state	スケールダウンプロセスの Pod の状態。この状態はデフォルトでは ACTIVE で、要求にサービスを提供することを意味します。	string	false

改訂日時: 2024-02-08