Red Hat JBoss Enterprise Application Platform 7.2

JBoss EAP for OpenShift Container Platform のスタートガイド

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

Last Updated: 2020-01-07

Red Hat JBoss Enterprise Application Platform 7.2 JBoss EAP for OpenShift Container Platform のスタートガイド

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

第1章 はじめに

1.1. RED HAT JBOSS ENTERPRISE APPLICATION PLATFORM (JBOSS EAP) とは

Red Hat JBoss Enterprise Application Platform 7 (JBoss EAP) は、オープン標準に構築されたミドルウェアプラットフォームで、Java Enterprise Edition 7 仕様に準拠します。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ース (CLI) により、XML 設定ファイルを編集する必要がなく、タスクをスクリプト化および自動化する機能が追加されます。さらに、JBoss EAP には、セキュアでスケーラブルな Java EE アプリケーションの迅速な開発、デプロイ、および実行を可能にする API と開発フレームワークが含まれています。JBoss EAP 7 は、Java EE 8 full および web プロファイル仕様の認定実装です。

1.2. OPENSHIFT での JBOSS EAP の仕組み

Red Hat は、OpenShift と使用するために設計された JBoss EAP のコンテナー化イメージを提供します。このイメージを使用すると、開発者はハイブリッド環境全体にデプロイされたアプリケーションを迅速かつ簡単にビルド、スケール、およびテストできます。

1.3. 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。管理 CLI はトラブルシューティングの目的で Pod 内からアクセスできます。
管理対象ドメイン	サポート対象外	JBoss EAP 管理対象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	一部サポート対象	JBoss EAP for OpenShift イメージでトランザクションリカバリーを実行するときに、一部 サポートされないトランザクションリカバリーシナリオおよび警告 があります。

1.4. バージョンの互換性とサポート

本ガイドでは、以下の JBoss EAP for OpenShift イメージを取り上げます。

- **jboss-eap-7/eap72-openshift**(JDK 8)
- **jboss-eap-7/eap72-openjdk11-openshift-rhel8**(JDK 11)

Red Hat Container Catalog でこれらのイメージの最新タグの情報を確認できます。

- JDK 8
- JDK 11

Hawkular エージェントは JDK 11 ではアクティブでなく、設定された場合は無視されます。

JBoss EAP for OpenShift は頻繁に更新されます。そのため、イメージのどのバージョンが OpenShift のどのバージョンと互換性があるかを理解することが重要になります。バージョンの互換性とサポートの詳細は、Red Hat カスタマーポータルの「OpenShift Container Platform 3.x のテスト済みインテグレーションを」参照してください。

1.4.1. OpenShift 4.1 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 4.1 Web コンソールで利用できなくなりました。

OpenShift Enterprise Administration (CL280) 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。現在では、この動作は安全ではないと見なされ、サポート対象外になりました。その結果、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。

1.5. テクノロジープレビューの機能

自動化トランザクションリカバリー

重要

この機能はテクノロジープレビューとしてのみ提供されます。テクノロジープレビューの機能は本番環境での使用はサポートされず、今後大きな変更がある場合があります。テクノロジープレビュー機能のサポート範囲については、Red Hat カスタマーポータルの「テクノロジプレビュー機能のサポート範囲」を参照してください。

クラスターがスケールダウンしたとき、トランザクションブランチがインダウトの状態になる可能性があります。JBoss EAP for OpenShift には、このようなブランチを完全にする 自動化トランザクションリカバリー機能 があります。現時点では、この実装の自動化トランザクションリカバリーはテクノロジープレビューとしてのみ提供されます。

アプリケーション Pod のスケールダウンで自動化トランザクションリカバリーを実証するために提供される **eap72-tx-recovery-s2i** アプリケーションテンプレートもテクノロジープレビューとしてのみ提供されます。

JDK 11 イメージストリームでは、アプリケーションテンプレートは **eap72-openjdk11-tx-recovery-s2i** です。

第2章 JBOSS EAP FOR OPENSHIFT イメージでの JAVA アプリケーションのビルドおよび実行

以下のワークフローでは、Source-to-Image (S2I) プロセスを使用して JBoss EAP for OpenShift イメージ上で Java アプリケーションをビルドおよび実行します。

たとえば、この手順では **kitchensink** クイックスタートが使用されます。これは JSF、CDI、EJB、JPA、および Bean Validation を使用して Java 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 7 に同梱される **kitchensink** クイックスタートを参照してください。

2.1. 要件

このワークフローでは、OpenShift インスタンスがすでにインストールされ、運用されていることを前提とします (OpenShift Primer の OpenShift インスタンスと同様)。

2.2. アプリケーションのデプロイメントに向けた OPENSHIFT の準備

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 で新しいプロジェクトを作成します。
   プロジェクトでは、1 つのユーザーグループが他のグループとは別にコンテンツを整理および管理することができます。以下のコマンドを使用すると OpenShift でプロジェクトを作成できます。

   ```
   $ 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** という名前の新規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、キーストアとシークレットの作成が必要になります。たとえば、**eap72-https-s2i** テンプレートを使用している場合、キーストアとシークレットを作成する必要があります。

   kitchensink クイックスタートのこのワークフローは、HTTPS テンプレートを使用しないため、キーストアとシークレットは必要ありません。

   a. キーストアを作成します。

警告

以下のコマンドは自己署名証明書を生成しますが、本番環境では信用性が確認された認証局 (CA) から購入した独自の SSL 証明書を SSL で暗号化された接続 (HTTPS) に使用することが推奨されます。

以下のように、Java **keytool** コマンドを使用して、キーストアを生成することができます。

```
$ 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. キーストアからシークレットを作成します。
以下のコマンドを使用して、作成したキーストアからシークレットを作成します。

```
$ oc secrets new SECRET_NAME 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 secrets new eap7-app-secret keystore.jks
```

2.3. RED HAT コンテナーレジストリーへの認証の設定

JBoss EAP for OpenShift イメージをインポートおよび使用する前に、最初に Red Hat コンテナーレジストリーへの認証を設定する必要があります。

Red Hat は、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、Red Hat コンテナーレジストリーへのアクセスを設定することを推奨します。こうすると、お持ちの Red Hat アカウントのユーザー名やパスワードを OpenShift 設定に使用または保存する必要がありません。

1. Red Hat カスタマーポータルの手順にしたがって、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ます。
2. トークンの OpenShift シークレットが含まれる YAML ファイルをダウンロードします。YAML ファイルは、トークンの **Token Information** ページの **OpenShift Secret** タブからダウンロードできます。
3. ダウンロードした YAML ファイルを使用して、OpenShift プロジェクトの認証トークンシークレットを作成します。

```
oc create -f 1234567_myserviceaccount-secret.yaml
```

4. 以下のコマンドを使用して、OpenShift プロジェクトのシークレットを設定します。シークレット名は前のステップで作成したシークレットの名前に置き換えてください。

```
oc secrets link default 1234567-myserviceaccount-pull-secret --for=pull
oc secrets link builder 1234567-myserviceaccount-pull-secret --for=pull
```

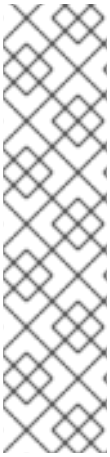
セキュリティー保護されたレジストリーへのアクセス設定方法については、OpenShift ドキュメントを参照してください。

Red Hat コンテナーレジストリーへの認証の設定 に関する詳細は、Red Hat カスタマーポータルを参照してください。

2.4. 最新の JBOSS EAP FOR OPENSHIFT イメージストリームおよびテンプレートのインポート

以下のコマンドを使用して、最新の JBoss EAP for OpenShift のイメージストリームおよびテンプレートを OpenShift プロジェクトの namespace にインポートします。

```
for resource in \
  eap72-image-stream.json \
  eap72-amq-persistent-s2i.json \
  eap72-amq-s2i.json \
  eap72-basic-s2i.json \
  eap72-https-s2i.json \
  eap72-mongodb-persistent-s2i.json \
  eap72-mongodb-s2i.json \
  eap72-mysql-persistent-s2i.json \
  eap72-mysql-s2i.json \
  eap72-postgresql-persistent-s2i.json \
  eap72-postgresql-s2i.json \
  eap72-third-party-db-s2i.json \
  eap72-tx-recovery-s2i.json \
  eap72-sso-s2i.json
do
  oc replace --force -f \
https://raw.githubusercontent.com/jboss-container-images/jboss-eap-7-openshift-image/eap72/templates/${resource}
done
```



注記

上記のコマンドを使用してインポートされた JBoss EAP イメージストリームおよびテンプレートは、OpenShift プロジェクト内のみで利用できます。

一般的な **openshift** namespace にアクセスできる管理者権限を持っている場合、すべてのプロジェクトがイメージストリームおよびテンプレートにアクセスできるようにするには、コマンドの **oc replace** 行に **-n openshift** を追加します。例を以下に示します。

```
...
oc replace -n openshift --force -f \
...
```

2.5. JBOSS EAP S2I (SOURCE-TO-IMAGE) アプリケーションの OPENSHIFT へのデプロイ

1. JBoss EAP for OpenShift イメージと Java アプリケーションのソースコードを使用して、新しい OpenShift アプリケーションを作成します。Red Hat は、提供される JBoss EAP for OpenShift テンプレートの 1 つを S2I ビルドに使用することを推奨します。
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、「**アプリケーションのデプロイメントに向けた OpenShift の準備**」で GitHub の **kitchensink** ソースコードを使用して作成した eap-demo プロジェクトに **eap72-basic-s2i** テンプレートを使用します。

```
oc new-app --template=eap72-basic-s2i \ 1
 -p IMAGE_STREAM_NAMESPACE=eap-demo \ 2
 -p SOURCE_REPOSITORY_URL=https://github.com/jboss-developer/jboss-eap-quickstarts
\ 3
 -p SOURCE_REPOSITORY_REF=openshift \ 4
 -p CONTEXT_DIR=kitchensink 5
```

1. 使用するテンプレート。
2. 最新のイメージとテンプレートは、プロジェクトの namespace にインポートされたため、イメージストリームが見つかる場所の namespace を指定する必要があります。通常はプロジェクトの名前になります。
3. アプリケーションのソースコードが含まれるリポジトリーの URL。
4. ソースコードに使用する Git リポジトリー参照。Git ブランチやタグ参照にすることができます。
5. ビルドするソースリポジトリー内のディレクトリー。

注記

テンプレートは、多くのテンプレートパラメーターにデフォルト値を指定でき、一部またはすべてのデフォルトをオーバーライドする必要がある場合があります。パラメーターのリストやデフォルト値などのテンプレートの情報を表示するには、コマンド **oc describe template TEMPLATE_NAME** を使用します。

注記

新しい OpenShift アプリケーションを作成するときに、環境変数を設定 することもあります。

たとえば、**eap72-https-s2i** などの HTTPS テンプレートを使用している場合は、必要な HTTPS 環境変数 である **HTTPS_NAME**、**HTTPS_PASSWORD**、および **HTTPS_KEYSTORE** を指定し、キーストアの詳細と一致するようにする必要があります。

2. ビルド設定の名前を取得します。

```
$ oc get bc -o name
```

3. 取得したビルド設定の名前を使用し、Maven のビルドの進捗を表示します。

```
-
```

```
$ oc logs -f buildconfig/BUILD_CONFIG_NAME
```

たとえば、**kitchensink** クイックスタートでは、以下のコマンドで Maven ビルドの進捗を表示します。

```
$ oc logs -f buildconfig/eap-app
```

2.6. デプロイメント後のタスク

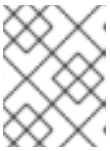
アプリケーションによっては、OpenShift アプリケーションのビルドおよびデプロイ後に一部のタスクを実行する必要が場合があります。これには、サービスを公開して OpenShift の外部からアプリケーションを閲覧可能にする作業や、アプリケーションを特定数のレプリカにスケーリングする作業などが含まれることがあります。

1. 以下のコマンドを使用してアプリケーションのサービス名を取得します。

```
$ oc get service
```

2. メインサービスをルートとして公開し、OpenShift 外部からアプリケーションにアクセスできるようにします。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必要なサービスとポートを公開します。

```
$ oc expose service/eap-app --port=8080
```



注記

テンプレートを使用してアプリケーションを作成した場合は、ルートがすでに存在することがあります。存在する場合は次のステップに進みます。

3. ルートの URL を取得します。

```
$ oc get route
```

4. この URL を使用して web ブラウザーでアプリケーションにアクセスします。URL は前のコマンド出力にある **HOST/PORT** フィールドの値になります。
アプリケーションが JBoss EAP ルートコンテキストを使用しない場合、アプリケーションのコンテキストを URL に追加します。たとえば、**kitchensink** クイックスタートでは、URL は **http://HOST_PORT_VALUE/kitchensink/** のようになります。

5. 任意で、以下のコマンドを実行してアプリケーションインスタンスをスケールアップすることもできます。これは、レプリカの数を **3** に増やします。

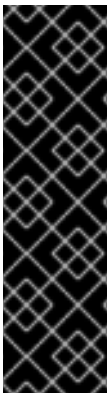
```
$ oc scale deploymentconfig DEPLOYMENTCONFIG_NAME --replicas=3
```

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アプリケーションをスケールアップします。

```
$ oc scale deploymentconfig eap-app --replicas=3
```

第3章 JAVA アプリケーションに対して JBOSS EAP FOR OPENSHIFT イメージを設定

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。OpenShift S2I プロセスをアプリケーションテンプレートパラメーターと環境変数とともに使用する方法が推奨されます。



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。

OpenShift S2I プロセスをアプリケーションテンプレートパラメーターと環境変数とともに使用して、JBoss EAP for OpenShift イメージ内部の JBoss EAP インスタンスの設定を変更することが強く推奨されます。

3.1. JBOSS EAP FOR OPENSHIFT の S2I プロセスの仕組み

注記

変数 **EAP_HOME** を使用して、JBoss EAP for OpenShift イメージ内部の JBoss EAP インストールへのパスを示します。

JBoss EAP for OpenShift の S2I プロセスは以下のように動作します。

1. **pom.xml** ファイルがソースコードリポジトリーに存在する場合、Maven のビルドプロセスは **$MAVEN_ARGS** 環境変数のコンテンツを使用するようトリガーされます。**$MAVEN_ARGS** 環境変数でカスタム Maven 引数またはオプションを指定することは可能ですが、Red Hat は **$MAVEN_ARGS_APPEND** 環境変数を使用して指定することを推奨します。**$MAVEN_ARGS_APPEND** 変数は **$MAVEN_ARGS** からデフォルトの引数を取り、**$MAVEN_ARGS_APPEND** からのオプションを追加します。

   デフォルトでは、OpenShift プロファイルは Maven の **package** ゴールを使用します。 これには、テストをスキップするシステムプロパティー (**-DskipTests**) や Red Hat GA リポジトリーを有効にするシステムプロパティー (**-Dcom.redhat.xpaas.repo**) が含まれます。

   注記

   JBoss EAP for OpenShift イメージのプロキシーの背後で Maven を使用するには、**$HTTP_PROXY_HOST** および **$HTTP_PROXY_PORT** 環境変数を設定します。任意で、**$HTTP_PROXY_USERNAME**、**HTTP_PROXY_PASSWORD**、および **HTTP_PROXY_NONPROXYHOSTS** 変数を設定することもできます。

2. 成功した Maven ビルドの結果は、JBoss EAP for OpenShift イメージ内の **EAP_HOME/standalone/deployments/** ディレクトリーにコピーされます。これには、**$ARTIFACT_DIR** 環境変数によって指定されたソースリポジトリーからの JAR、WAR、および EAR ファイルがすべて含まれます。**$ARTIFACT_DIR** のデフォルト値は Maven のターゲットディレクトリーです。

3. **configuration** ソースリポジトリーディレクトリーのすべてのファイルは、JBoss EAP for

OpenShift イメージ内の **EAP_HOME/standalone/configuration/** ディレクトリーにコピーされます。カスタムの JBoss EAP 設定ファイルを使用する場合、ファイル名を **standalone-openshift.xml** にする必要があります。

4. **modules** ソースリポジトリーディレクトリーのすべてのファイルは、JBoss EAP for OpenShift イメージ内の **EAP_HOME/modules/** ディレクトリーにコピーされます。

S2I プロセスを指示してカスタム Maven アーティファクトリポジトリーミラーを利用する方法の追加情報は「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。

3.2. 環境変数を使用した JBOSS EAP FOR OPENSHIFT の設定

JBoss EAP for OpenShift イメージを設定する方法として、環境変数の使用が推奨されます。アプリケーションコンテナーおよびビルドコンテナーに 環境変数を指定 する方法については、OpenShift ドキュメントを参照してください。

たとえば、OpenShift アプリケーションの作成時に、環境変数を使用して JBoss EAP インスタンスの管理ユーザー名およびパスワードを設定することができます。

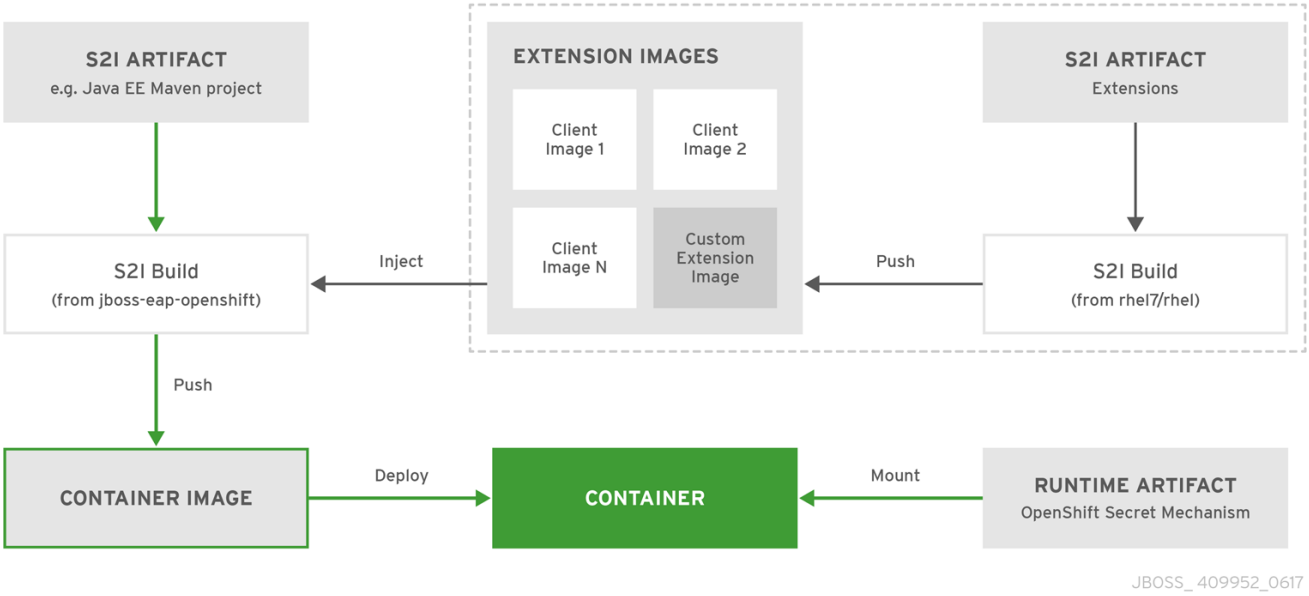
```
oc new-app --template=eap72-basic-s2i \
 -p IMAGE_STREAM_NAMESPACE=eap-demo \
 -p SOURCE_REPOSITORY_URL=https://github.com/jboss-developer/jboss-eap-quickstarts \
 -p SOURCE_REPOSITORY_REF=openshift \
 -p CONTEXT_DIR=kitchensink \
 -e ADMIN_USERNAME=myspecialuser \
 -e ADMIN_PASSWORD=myspecialp@ssw0rd
```

JBoss EAP for OpenShift イメージの利用可能な環境変数は、「参考情報」のリストを参照してください。

3.3. ビルド拡張およびプロジェクトアーティファクト

JBoss EAP for OpenShift イメージは、さまざまなアーティファクトを使用して OpenShift のデータベースサポートを拡張します。これらのアーティファクトは異なるメカニズムを介してビルドイメージに含まれます。

- S2I プロセスの間にイメージにインジェクトされる S2I アーティファクト。
- OpenShift シークレットメカニズムを介して提供される環境ファイルからの ランタイムアーティファクト。



重要

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、JDK 8 イメージストリームでは非推奨になりました。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が推奨されます。

以下の内部データソースは、JBoss EAP for OpenShift の JDK 11 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。

3.3.1. S2I アーティファクト

S2I アーティファクトには、モジュール、ドライバー、およびデプロイメントに必要な設定インフラストラクチャーを提供する追加の汎用デプロイメントが含まれます。この設定は S2I プロセスの間にイメージに組み込まれるため、データソースと関連するリソースアダプターのみをランタイムに設定する必要があります。

S2I プロセスを指示してカスタム Maven アーティファクトリポジトリーミラーを利用する方法の追加情報は「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。

3.3.1.1. モジュール、ドライバー、および汎用デプロイメント

JBoss EAP for OpenShift イメージにこれらの S2I アーティファクトが含まれるようにする方法はいくつかあります。

1. アプリケーションソースデプロイメントディレクトリーにアーティファクトが含まれるようにします。アーティファクトはビルド中にダウンロードされ、イメージにインジェクトされます。これは、JBoss EAP for OpenShift イメージでアプリケーションをデプロイするのと似てい

ます。

2. **CUSTOM_INSTALL_DIRECTORIES** 環境変数が含まれるようにします。これは、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です。S2I プロセスにこの情報が含まれるようにする方法は 2 つあります。

   - 指定されたインストールディレクトリーの **install.sh** スクリプト。インストールスクリプトは S2I プロセス中に実行され、問題なく動作します。

     install.sh スクリプトの例

     ```
     #!/bin/bash

     injected_dir=$1
     source /usr/local/s2i/install-common.sh
     install_deployments ${injected_dir}/injected-deployments.war
     install_modules ${injected_dir}/modules
     configure_drivers ${injected_dir}/drivers.env
     ```

     install.sh スクリプトの役割は、**install-common.sh** によって提供される API を使用してベースイメージをカスタマイズすることです。 **install-common.sh** には、モジュール、ドライバー、および汎用デプロイメントをインストールおよび設定するために **install.sh** スクリプトによって使用される関数が含まれます。

     install-common.sh 内に含まれる関数は次のとおりです。

     - **install_modules**
     - **configure_drivers**
     - **install_deployments**

       モジュール

       モジュールは、クラスローディングおよび依存関係管理に使用されるクラスの論理グループです。モジュールは、アプリケーションサーバーの **EAP_HOME/modules/** ディレクトリーに定義されます。各モジュールは、**EAP_HOME/modules/org/apache/** のようにサブディレクトリーとして存在します。各モジュールのディレクトリーには、デフォルトが main であるスロットサブディレクトリーが含まれ、**module.xml** 設定ファイルと必要な JAR ファイルすべてが含まれます。

       MySQL および PostgreSQL JDBC ドライバーの **module.xml** ファイルの設定に関する詳細は、JBoss EAP『設定ガイド』の「データソース設定の例」を参照してください。

       module.xml ファイルの例

       ```
       <?xml version="1.0" encoding="UTF-8"?>
       <module xmlns="urn:jboss:module:1.0" name="org.apache.derby">
          <resources>
             <resource-root path="derby-10.12.1.1.jar"/>
             <resource-root path="derbyclient-10.12.1.1.jar"/>
          </resources>
          <dependencies>
             <module name="javax.api"/>
       ```

```
        <module name="javax.transaction.api"/>
    </dependencies>
</module>
```

PostgreSQL データソースの module.xml ファイルの例

```
<?xml version="1.0" encoding="UTF-8"?>
<module xmlns="urn:jboss:module:1.0" name="org.postgresql">
<resources>
<resource-root path="postgresql-jdbc.jar"/>
</resources>
<dependencies>
<module name="javax.api"/>
<module name="javax.transaction.api"/>
</dependencies>
</module>
```

MySQL Connect/J 8 データソースの module.xml ファイルの例

```
<?xml version="1.0" encoding="UTF-8"?>
<module xmlns="urn:jboss:module:1.0" name="com.mysql">
<resources>
<resource-root path="mysql-connector-java-8.0.Z.jar" />
</resources>
<dependencies>
<module name="javax.api"/>
<module name="javax.transaction.api"/>
</dependencies>
</module>
```

注記

mysql-connector-java-8.0.Z.jar の「.Z」はダウンロードした **JAR** ファイルのバージョンを示します。ファイル名は変更できますが、名前は **module.xml** ファイルの名前に一致する必要があります。

install.sh の **install_modules** 関数は、**module.xml** とともに該当の JAR ファイルを JBoss EAP の modules ディレクトリーにコピーします。

ドライバー

ドライバーはモジュールとしてインストールされます。ドライバーは **configure_drivers** 関数によって **install.sh** に設定されます。 この設定プロパティーは ランタイムアーティファクト 環境ファイルに定義されます。

drivers.env ファイルの例

```
#DRIVER
DRIVERS=DERBY
DERBY_DRIVER_NAME=derby
DERBY_DRIVER_MODULE=org.apache.derby
DERBY_DRIVER_CLASS=org.apache.derby.jdbc.EmbeddedDriver
DERBY_XA_DATASOURCE_CLASS=org.apache.derby.jdbc.EmbeddedXADataSourc
e
```

内部データソースドライバーの追加

MySQL および PostgreSQL データソースは、事前に設定された内部データソースとして提供されなくなりました。ただし、これらのドライバーは、「モジュール、ドライバー、および汎用デプロイメント」で説明されているように、モジュールとしてインストールすることができます。

メカニズムは **Derby** ドライバーの例にならい、S2I アーティファクトを使用します。インストールする各データソースの **drivers.env** ファイルを作成します。

MySQL データソースの drivers.env ファイルの例

```
#DRIVER
DRIVERS=MYSQL
MYSQL_DRIVER_NAME=mysql
MYSQL_DRIVER_MODULE=org.mysql
MYSQL_DRIVER_CLASS=com.mysql.cj.jdbc.Driver
MYSQL_XA_DATASOURCE_CLASS=com.mysql.jdbc.jdbc2.optional.MysqlXADataSou
rce
```

PostgreSQL データソースの drivers.env ファイルの例

```
#DRIVER
DRIVERS=POSTGRES
POSTGRES_DRIVER_NAME=postgres
POSTGRES_DRIVER_MODULE=org.postgresql
POSTGRES_DRIVER_CLASS=org.postgresql.Driver
POSTGRES_XA_DATASOURCE_CLASS=org.postgresql.xa.PGXADataSource
```

MySQL や PostgreSQL など、さまざまなドライバーのダウンロード場所に関する情報は、『設定ガイド』の「 JDBC ドライバーのダウンロード場所 」を参照してください。

汎用デプロイメント

JAR、WAR、RAR、EAR などのデプロイ可能なアーカイブファイルは、**install-common.sh** の API によって提供される **install_deployments** を使用して、インジェクトされたイメージからデプロイすることができます。

- **CUSTOM_INSTALL_DIRECTORIES** 環境変数が宣言されていても、カスタムインストールディレクトリーに **install.sh** スクリプトがない場合、以下のアーティファクトディレクトリーがビルドイメージの該当する場所にコピーされます。
  - **modules/*** は **$JBOSS_HOME/modules/system/layers/openshift** にコピーされます。
  - **configuration/*** は **$JBOSS_HOME/standalone/configuration** にコピーされます。
  - **deployments/*** は **$JBOSS_HOME/standalone/deployments** にコピーされます。

  これは **install.sh** の代替方法と比べ基本的な設定方法となります。 アーティファクトが適切に構築される必要があります。

3.3.2. ランタイムアーティファクト

3.3.2.1. データソース

データソースの種類は 3 つあります。

1. デフォルトの内部データソース。これらは PostgreSQL、MySQL、および MongoDB になります。これらのデータソースは、Red Hat レジストリー介して OpenShift ではデフォルトで利用でき、JDK 8 イメージストリームに追加の環境ファイルを設定する必要がありません。DB_SERVICE_PREFIX_MAPPING 環境変数を、検出してデータソースとして使用する OpenShift サービスの名前に設定します。

2. その他の内部データソース。これらは、Red Hat レジストリーを介してデフォルトでは利用できない、OpenShift 上で実行されるデータソースです。これらのデータソースの設定は、OpenShift のシークレットに追加された環境ファイルによって提供されます。

3. OpenShift 上では実行されない外部データソース。外部データソースの設定は、OpenShift のシークレットに追加された環境ファイルによって提供されます。

例: データソース環境ファイル

```
# derby datasource
ACCOUNTS_DERBY_DATABASE=accounts
ACCOUNTS_DERBY_JNDI=java:/accounts-ds
ACCOUNTS_DERBY_DRIVER=derby
ACCOUNTS_DERBY_USERNAME=derby
ACCOUNTS_DERBY_PASSWORD=derby
ACCOUNTS_DERBY_TX_ISOLATION=TRANSACTION_READ_UNCOMMITTED
ACCOUNTS_DERBY_JTA=true

# Connection info for xa datasource
ACCOUNTS_DERBY_XA_CONNECTION_PROPERTY_DatabaseName=/home/jboss/source/data/databases/derby/accounts

# _HOST and _PORT are required, but not used
ACCOUNTS_DERBY_SERVICE_HOST=dummy
ACCOUNTS_DERBY_SERVICE_PORT=1527
```

DATASOURCES プロパティーは、データソースプロパティー接頭辞のコンマ区切りリストです。これらの接頭辞は、データソースのすべてのプロパティーに追加されます。複数のデータソースを 1 つの環境ファイルに含むことができます。また、各データソースを個別の環境ファイルに提供することもできます。

データソースには、接続プール固有のプロパティーとデータベースドライバー固有のプロパティーの 2 種類のプロパティーが含まれます。データベースドライバー固有のプロパティーは、ドライバー自体がドライバー S2I アーティファクトとして設定されるため、汎用の **XA_CONNECTION_PROPERTY** を使用します。ドライバープロパティーの接尾辞は、データソースの特定のドライバーに固有します。

上記の例では、**ACCOUNTS** はデータソース接頭辞、**XA_CONNECTION_PROPERTY** は汎用ドライバープロパティー、**DatabaseName** はドライバー固有のプロパティーになります。

データソース環境ファイルは、プロジェクトの OpenShift シークレットに追加されます。これらの環境ファイルは、**ENV_FILES** 環境プロパティーを使用して、テンプレート内で呼び出されます。 この環境プロパティーの値は、以下のような完全修飾環境ファイルのコンマ区切りリストです。

```
{
    “Name”: “ENV_FILES”,
    “Value”: “/etc/extensions/datasources1.env,/etc/extensions/datasources2.env”
```

```
}
```

3.3.2.2. リソースアダプター

リソースアダプターの設定は、OpenShift のシークレットに追加された環境ファイルによって提供されます。

表3.1 リソースアダプタープロパティー

属性	説明
PREFIX_ID	サーバー設定ファイルに指定されたリソースアダプターの識別子。
PREFIX_ARCHIVE	リソースアダプターアーカイブ。
PREFIX_MODULE_SLOT	**module.xml** 設定ファイルと必要な JAR ファイルがすべて含まれるスロットサブディレクトリー。
PREFIX_MODULE_ID	オブジェクトファクトリー Java クラスをロードできる JBoss モジュール ID。
PREFIX_CONNECTION_CLASS	管理された接続ファクトリーまたは管理オブジェクトの完全修飾クラス名。
PREFIX_CONNECTION_JNDI	接続ファクトリーの JNDI 名。
PREFIX_PROPERTY_ParentDirectory	データファイルが格納されるディレクトリー。
PREFIX_PROPERTY_AllowParentPaths	**AllowParentPaths** を **false** に設定して、パスの **..** を許可しないようにします。これにより、親ディレクトリーに含まれていないファイルを要求しないようにします。
PREFIX_POOL_MAX_SIZE	プールの最大接続数。各サブプールではこの値を超える接続は作成されません。
PREFIX_POOL_MIN_SIZE	プールの最小接続数。
PREFIX_POOL_PREFILL	プールをプレフィルすべきかどうかを指定します。値の変更後にサーバーを再起動する必要があります。
PREFIX_POOL_FLUSH_STRATEGY	エラーの場合にプールがどのようにフラッシュされるか。有効な値は **FailingConnectionOnly** (デフォルト)、**IdleConnections**、および **EntirePool** です。

RESOURCE_ADAPTERS プロパティーは、リソースアダプタープロパティー接頭辞のコンマ区切りリストです。接頭辞はそのリソースアダプターのすべてのプロパティーに追加されます。複数のリソースアダプターを 1 つの環境ファイルに含めることができます。以下の例では、**MYRA** がリソースアダプターの接尾辞として使用されます。各リソースアダプターを個別の環境ファイルに提供することもできます。

例: リソースアダプター環境ファイル

```
#RESOURCE_ADAPTER
RESOURCE_ADAPTERS=MYRA
MYRA_ID=myra
MYRA_ARCHIVE=myra.rar
MYRA_CONNECTION_CLASS=org.javaee7.jca.connector.simple.connector.outbound.MyManagedConnectionFactory
MYRA_CONNECTION_JNDI=java:/eis/MySimpleMFC
```

リソースアダプター環境ファイルは、プロジェクト namespace の OpenShift シークレットに追加されます。これらの環境ファイルは、**ENV_FILES** 環境プロパティーを使用して、テンプレート内で呼び出されます。 この環境プロパティーの値は、以下のような完全修飾環境ファイルのコンマ区切りリストです。

```
{
    "Name": "ENV_FILES",
    "Value": "/etc/extensions/resourceadapter1.env,/etc/extensions/resourceadapter2.env"
}
```

3.4. JBOSS EAP FOR OPENSHIFT イメージのデプロイメントに関する考慮事項

3.4.1. スケールアップおよび永続ストレージのパーティショニング

永続ストレージを用いて JBoss EAP をデプロイする方法は、単一ノードのパーティショニングと複数ノードのパーティショニングの 2 つがあります。

単一ノードのパーティショニングは、トランザクションデータを含む JBoss EAP データストアディレクトリーをストレージボリュームに格納します。

複数ノードのパーティショニングは、追加の独立した **split-n** ディレクトリーを作成し、各 JBoss EAP Pod のトランザクションデータを格納します。**n** は、増分整数です。JBoss EAP Pod が更新された場合、予期せず停止した場合、または再デプロイされた場合、この通信は変更されません。JBoss EAP Pod が再度操作可能になると、関連する split ディレクトリーに再接続し、以前と同様に続行されます。新しい JBoss EAP Pod が追加されると、対応する **split-n** ディレクトリーがその Pod に作成されます。

複数ノードの設定を有効にするには、**SPLIT_DATA** パラメーターを **true** に設定する必要があります。これにより、データストアとして使用される永続ボリューム内の各インスタンスに対して、サーバーが独立した **split-n** ディレクトリーを作成します。

現在、これが **eap72-tx-recovery-s2i** テンプレートのデフォルト設定です。

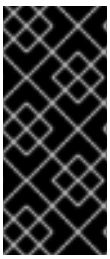
重要

単一ノードと複数ノードのパーティショニングではストレージの方法が異なるため、デプロイメントを単一ノードから複数ノードに変更すると、data ディレクトリーにこれまで保存されたすべてのデータ (メッセージやトランザクションログを含む) がアプリケーションから失われます。ストレージパスが一致しないため、デプロイメントを複数ノードから単一ノードに変更した場合でも同様です。

3.4.2. スケールダウンおよびトランザクションリカバリー

JBoss EAP for OpenShift イメージが 複数ノード 設定を使用してデプロイされると、クラスターがスケールダウンした場合に、終了する Pod の data ディレクトリーに予期せず終了されたトランザクションが残される可能性があります。

クラスターが次回スケールアップするまで、終了する Pod のデータストア内にトランザクションが残らないようにするため、**eap72-tx-recovery-s2i** JBoss EAP テンプレートは、トランザクションの移行を管理するマイグレーション Pod が含まれる 2 つ目のデプロイメントを作成します。マイグレーション Pod は、JBoss EAP 永続ボリューム内の独立した各 **split-n** ディレクトリーをスキャンし、終了する Pod と関連するデータストアを特定し、終了する Pod のすべてのトランザクションが完了するまで実行を継続します。



重要

永続ボリュームは JBoss EAP アプリケーション Pod と移行 Pod の両方によって読み書きモードでアクセスする必要があるため、**ReadWriteMany** アクセスモードで作成する必要があります。このアクセスモードは、**GlusterFS** および **NFS** プラグインを使用する永続ボリュームでのみサポートされています。詳細は、**永続ボリュームのサポート対象アクセスモード表**を参照してください。

詳細情報は、クラスターをスケールダウンするときの JBoss EAP for OpenShift イメージの自動化トランザクションリカバリー機能を実行する「ワークフローの例: クラスターをスケールダウンするときの自動化トランザクションリカバリー機能」を参照してください。

第4章 JBOSS EAP FOR OPENSHIFT の JDK 11 イメージへの移行

4.1. JDK 11 イメージを使用したアプリケーションのデプロイメントに向けた OPENSHIFT の準備

JDK 11 イメージを使用してアプリケーションのデプロイメントに向けて OpenShift を準備するには、「アプリケーションのデプロイメントに向けた OpenShift の準備」と同じ手順にしたがいます。

4.2. JDK 11 イメージのインポート

以下のコマンドを使用して、JBoss EAP for OpenShift の JDK 11 イメージストリームおよびテンプレートを OpenShift プロジェクトの namespace にインポートします。

```
for resource in \
  eap72-openjdk11-image-stream.json \
  eap72-openjdk11-amq-persistent-s2i.json \
  eap72-openjdk11-amq-s2i.json \
  eap72-openjdk11-basic-s2i.json \
  eap72-openjdk11-https-s2i.json \
  eap72-openjdk11-sso-s2i.json \
  eap72-openjdk11-starter-s2i.json \
  eap72-openjdk11-third-party-db-s2i.json \
  eap72-openjdk11-tx-recovery-s2i.json
do
 oc replace --force -f \
https://raw.githubusercontent.com/jboss-container-images/jboss-eap-7-openshift-image/eap72-openjdk11-ubi8/templates/${resource}
done
```

重要

以下の内部データソースおよびドライバーは、JBoss EAP for OpenShift の JDK 11 イメージとは提供されません。

- MySQL
- PostgreSQL
- MongoDB

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が推奨されます。

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。

4.3. JDK 11 イメージを使用した JBOSS EAP S2I アプリケーションの OPENSHIFT へのデプロイ

JBoss EAP の S2I アプリケーションを OpenShift にデプロイするには、「JBoss EAP S2I (Source-to-Image) アプリケーションの OpenShift へのデプロイ」と同じ手順にしたがいます。

JDK 11 ストリームは、JDK 8 で使用される **eap72-basic-s2i** テンプレートの代わりに **eap72-openjdk11-basic-s2i** を S2I ビルドに使用します。

kitchensink クイックスタートをデプロイするには、以下のコマンドを使用して **eap72-openjdk11-basic-s2i** テンプレートを GitHub の **kitchensink** ソースコードと使用します。

```
oc new-app --template=eap72-openjdk11-basic-s2i \
 -p IMAGE_STREAM_NAMESPACE=eap-demo \
 -p SOURCE_REPOSITORY_URL=https://github.com/jboss-developer/jboss-eap-quickstarts.git \
 -p SOURCE_REPOSITORY_REF=openshift \
 -p CONTEXT_DIR=kitchensink \
```

eap-demo プロジェクトの **eap72-openjdk11-basic-s2i** テンプレートは「アプリケーションのデプロイメントに向けた OpenShift の準備」で作成済みです。

4.4. JDK 11 イメージに環境変数を使用した JBOSS EAP FOR OPENSHIFT の設定

JDK 11 イメージに環境変数を使用して JBoss EAP for OpenShift を設定するには、「環境変数を使用した JBoss EAP for OpenShift の設定」と同じ手順にしたがいます。

JDK 11 イメージストリームは、JDK 8 で使用される **eap72-basic-s2i** テンプレートの代わりに **eap72-openjdk11-basic-s2i** テンプレートを使用します。

環境変数を使用して JBoss EAP インスタンスの管理ユーザー名およびパスワードを設定するには、OpenShift アプリケーションの作成時に以下のコマンドを使用します。

```
oc new-app --template=eap72-openjdk11-basic-s2i \
-p IMAGE_STREAM_NAMESPACE=eap-demo \
-p SOURCE_REPOSITORY_URL=https://github.com/jboss-developer/jboss-eap-quickstarts \
-p SOURCE_REPOSITORY_REF=openshift \
-p CONTEXT_DIR=kitchensink \
-e ADMIN_USERNAME=myspecialuser \
-e ADMIN_PASSWORD=myspecialp@ssw0rd
```

第5章 アプリケーションの OPENSHIFT 4 への移行

5.1. OPENSHIFT 4 の LIVENESS および READINESS プローブ設定の更新

プローブの **YAML** 設定は、OpenShift 4 に移行する際に調整される必要があります。

OpenShift 3.11 では、Liveness プローブのデフォルト **YAML** 設定は以下のコードの例に似ています。

OpenShift 3.11 liveness プローブの YAML 設定サンプル

```
livenessProbe:
        exec:
          command:
            - /bin/bash
            - '-c'
            - /opt/eap/bin/livenessProbe.sh
        initialDelaySeconds: 60
        periodSeconds: 10
        successThreshold: 1
        failureThreshold: 3
```

この例では、Liveness プローブは JBoss EAP イメージ内の **/opt/eap/bin/livenessProbe.sh** にあります。プローブは、60 秒の最初の遅延後にトリガーされ、JBoss EAP サーバーで Pod が開始された後は 10 秒ごとにトリガーされます。

プローブは、**livenessProbe.sh** スクリプトの呼び出し試行が 3 回行われると失敗とみなされます。コンテナーは正常ではないとみなされ、OpenShift はそれぞれの Pod で JBoss EAP コンテナーを再起動します。

OpenShift 3.11 では、成功または失敗が決まるまで、1 回の呼び出しが 5 秒間続きます。OpenShift 4 では、1 回の呼び出しが 1 秒未満になります。

OpenShift 3.11 では、プローブの呼び出しが 5 秒間続き、10 秒間待機します。つまり、JBoss EAP イメージが正常ではない場合、Pod 内のコンテナーが再起動するまで、3 回にわたる呼び出しが約 35 秒間続きます。

OpenShift 4、3 では、呼び出しが 23 秒間続きます。OpenShift 4 のプローブの設定は、以下のように **YAML** 設定で調整する必要があります。

OpenShift 4 liveness プローブの YAML 設定サンプル

```
livenessProbe:
        exec:
          command:
            - /bin/bash
            - '-c'
            - /opt/eap/bin/livenessProbe.sh
        initialDelaySeconds: 60
        periodSeconds: 16
        successThreshold: 1
        failureThreshold: 3
```

この例では、**periodSeconds** が 6 秒分増えています。これにより、最初の呼び出しが 1 秒続き、16 秒間にわたり待機します。3 回の呼び出しは、プローブの OpenShift 3.11 の動作とほぼ同等で 約 34 秒続きます。

Readiness プローブの場合、同様の調整を **YAML** 設定に対して行う必要があります。

OpenShift 4 Readiness プローブの YAML 設定サンプル

```
readinessProbe:
        exec:
          command:
            - /bin/bash
            - '-c'
            - /opt/eap/bin/readinessProbe.sh
        initialDelaySeconds: 10
        periodSeconds: 16
        successThreshold: 1
        failureThreshold: 3
```

その他のリソース

Liveness および Readiness プローブ

第6章 トラブルシューティング

6.1. POD 再起動の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。

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。

6.2. 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 の設定を変更する場合は「Java アプリケーションに対して JBoss EAP for OpenShift イメージを設定」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。

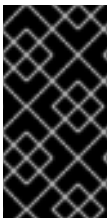
   ```
   $ oc rsh POD_NAME
   ```

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/eap/bin/jboss-cli.sh
```

第7章 上級者向けチュートリアル

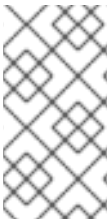
7.1. ワークフローの例: クラスターをスケールダウンするときの自動化トランザクションリカバリー機能



重要

この機能はテクノロジープレビューとしてのみ提供されます。テクノロジープレビューの機能は本番環境での使用はサポートされず、今後大きな変更がある場合があります。テクノロジープレビュー機能のサポート範囲については、Red Hat カスタマーポータルの「テクノロジプレビュー機能のサポート範囲」を参照してください。

このチュートリアルでは、クラスターをスケールダウンするときに JBoss EAP for OpenShift イメージの自動化トランザクションリカバリー機能を実行します。ここでは、**jta-crash-rec-eap7** クイックスタートサンプルと **eap72-tx-recovery-s2i** アプリケーションテンプレートは、クラスターのスケールダウンによって OpenShift Pod が終了されたときに XA トランザクションの問題が、どのように専用のマイグレーション Pod によってリカバリーされるかを示すために使用されます。



注記

jta-crash-rec-eap7 クイックスタートは、JBoss EAP に含まれる H2 データベースを使用します。これは、例でのみ使用される、ライトウェイトなリレーショナルデータソースのサンプルです。堅牢でもスケーラブルでもなく、サポートされないため、本番環境では使用しないでください。

7.1.1. デプロイメントの準備

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. 新しいプロジェクトを作成します。

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

3. 基盤の Pod の実行に使用される **default** サービスアカウントに view ロールを追加します。これにより、サービスアカウントが **eap-tx-demo** namespace のすべてのリソースを確認できるようになります。 これは、クラスターの管理に必要です。

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

4. 自動化トランザクションリカバリーが機能するには、JBoss EAP アプリケーションは **ReadWriteMany** 永続ボリュームを使用する必要があります。**${APPLICATION_NAME}-eap-claim 永続ボリュームクレーム** のデータを保持するため eap72-tx-recovery-s2i アプリケーションテンプレートによって想定される永続ボリュームのプロビジョニングを行います。

   この例は、以下の定義で NFS メソッドを使用してプロビジョニングされる永続ボリュームオブジェクトを使用します。

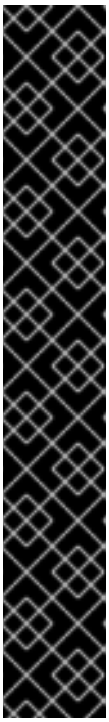
```
$ cat txpv.yaml
apiVersion: v1
kind: PersistentVolume
metadata:
```

```
  name: txpv
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteMany
  persistentVolumeReclaimPolicy: Retain
  nfs:
    path: /mnt/mountpoint
    server: 192.168.100.175
```

上記の定義の **path** および **server** フィールドを環境に合わせて更新し、以下のコマンドを使用して永続ボリュームのプロビジョニングを行います。

```
$ oc create -f txpv.yaml
persistentvolume "txpv" created
```

```
$ oc get pv
NAME      CAPACITY   ACCESSMODES   RECLAIMPOLICY   STATUS      CLAIM
STORAGECLASS   REASON    AGE
txpv      1Gi        RWX           Retain          Available                                   26s
```



重要

NFS メソッドを使用して **eap72-tx-recovery-s2i** アプリケーションテンプレートの永続ボリュームオブジェクトのプロビジョニングを行う場合、適切なパーミッションがある状態でマウントポイントをエクスポートするようにしてください。マウントポイントのエクスポート元となるホストで以下を実行します。

```
# chmod -R 777 /mnt/mountpoint
```

```
# cat /etc/exports
/mnt/mountpoint *(rw,sync,anonuid=185,anongid=185)
```

```
# exportfs -va
exporting *:/mnt/mountpoint
```

```
# setsebool -P virt_use_nfs 1
```

上記の **/mnt/mountpoint** パスは、環境に合わせて変更してください。

7.1.2. デプロイメント

1. eap72-tx-recovery-s2i アプリケーションテンプレートを使用して、**jta-crash-rec-eap7** クイックスタートをデプロイします。以下を指定します。

例: eap72-tx-recovery-s2i アプリケーションテンプレート

+

```
$ oc new-app --template=eap72-tx-recovery-s2i \
-p SOURCE_REPOSITORY_URL=https://github.com/jboss-openshift/openshift-quickstarts \
```

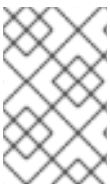
```
-p SOURCE_REPOSITORY_REF=master \
-p CONTEXT_DIR=jta-crash-rec-eap7 \
-e CUSTOM_INSTALL_DIRECTORIES=extensions/* \
--name=eap-app
--> Deploying template "openshift/eap72-tx-recovery-s2i" to project eap-tx-demo

    JBoss EAP 7.0 (tx recovery)
    ---------
    An example EAP 7 application. For more information about using this template, see
https://github.com/jboss-openshift/application-templates.

    A new EAP 7 based application has been created in your project.

    * With parameters:
       * Application Name=eap-app
       * Custom http Route Hostname=
       * Git Repository URL=https://github.com/jboss-openshift/openshift-quickstarts
       * Git Reference=master
       * Context Directory=jta-crash-rec-eap7
       * Queues=
       * Topics=
       * A-MQ cluster password=nyneOXUm # generated
       * Github Webhook Secret=PUW8Tmov # generated
       * Generic Webhook Secret=o7uD7qrG # generated
       * ImageStream Namespace=openshift
       * JGroups Cluster Password=MoR1Jthf # generated
       * Deploy Exploded Archives=false
       * Maven mirror URL=
       * ARTIFACT_DIR=
       * MEMORY_LIMIT=1Gi
       * EAP Volume Size=1Gi
       * Split the data directory?=true

--> Creating resources ...
   service "eap-app" created
   service "eap-app-ping" created
   route "eap-app" created
   imagestream "eap-app" created
   buildconfig "eap-app" created
   deploymentconfig "eap-app" created
   deploymentconfig "eap-app-migration" created
   persistentvolumeclaim "eap-app-eap-claim" created
--> Success
   Build scheduled, use 'oc logs -f bc/eap-app' to track its progress.
   Run 'oc status' to view your app.
```



注記

以上の例では、JDK 11 イメージストリームは JDK 8 イメージストリームで使用される **eap72-tx-recovery-s2i** の代わりに、**eap72-openjdk11-tx-recovery-s2i** アプリケーションテンプレートを使用します。

1. ビルドが完了するまで待ちます。**oc logs -f bc/eap-app** コマンドを使用すると、ビルドの状態を確認できます。

2. **JAVA_OPTS_APPEND** および **JBOSS_MODULES_SYSTEM_PKGS_APPEND** 環境変数の定義で **eap-app** デプロイメント設定を編集します。

```
$ oc get dc
NAME              REVISION   DESIRED   CURRENT   TRIGGERED BY
eap-app           1          1         1         config,image(eap-app:latest)
eap-app-migration  1         1         1         config,image(eap-app:latest)
```

```
$ oc set env dc/eap-app \
-e JBOSS_MODULES_SYSTEM_PKGS_APPEND="org.jboss.byteman" \
-e JAVA_OPTS_APPEND="-
javaagent:/tmp/src/extensions/byteman/byteman.jar=script:/tmp/src/src/main/scripts/xa.btm"
deploymentconfig "eap-app" updated
```

この設定は、以下のように XA トランザクションの処理を変更するよう、Byteman のトレースおよび監視ツールに通知します。

- 最初のトランザクションは成功するように常に許可されます。
- XA トラザクションが 2 つ目のトランザクションのフェーズ 2 を実行するとき、特定の Pod の JVM プロセスが停止されます。

7.1.3. JTA クラッシュリカバリーアプリケーションの使用

1. 現在の namespace で実行中の Pod をリストします。

```
$ oc get pods | grep Running
NAME                      READY     STATUS      RESTARTS   AGE
eap-app-2-r00gm           1/1       Running     0          1m
eap-app-migration-1-lvfdt  1/1       Running     0          2m
```

2. 新しい XA トラザクションを発行します。
   a. ブラウザーを開き、http://eap-app-eap-tx-demo.openshift.example.com/jboss-jta-crash-rec にアクセスして、アプリケーションを起動します。
   b. **Mercedes** を **Key** フィールド、**Benz** を **Value** フィールドに入力します。**Submit** ボタンをクリックします。
   c. しばらく待ち、**Refresh Table** リンクをクリックします。
   d. **Mercedes** エントリーが含まれるテーブル行がどのように **updated via JMS.** で更新されるか注意してください。 まだ更新されない場合は、**Refresh Table** リンクを数回クリックしてください。この代わりに、**eap-app-2-r00gm** Pod のログを調べ、トランザクションが適切に処理されたことを確認することもできます。

```
$ oc logs eap-app-2-r00gm | grep 'updated'
INFO  [org.jboss.as.quickstarts.xa.DbUpdaterMDB] (Thread-0 (ActiveMQ-client-global-threads-1566836606)) JTA Crash Record Quickstart: key value pair updated via JMS.
```

3. ブラウザーを使用し、http://eap-app-eap-tx-demo.openshift.example.com/jboss-jta-crash-rec で 2 つ目の XA トラザクションを発行します。
   a. **Land** を **Key** フィールドに、**Rover** を **Value** フィールドに入力します。**Submit** ボタンをクリックします。

b. しばらく待ち、**Refresh Table** リンクをクリックします。

c. **Land Rover** エントリーが **updated via ...** 接尾辞なしでどのように追加されたかに注目してください。

4. クラスターをスケールダウンします。

```
$ oc scale --replicas=0 dc/eap-app
deploymentconfig "eap-app" scaled
```

a. **eap-app-2-r00gm** Pod の終了がどのようにスケジュールされたかを確認します。

```
$ oc get pods
NAME                       READY     STATUS        RESTARTS   AGE
eap-app-1-build            0/1       Completed     0          4m
eap-app-2-r00gm            1/1       Terminating   0          2m
eap-app-migration-1-lvfdt  1/1       Running       0          3m
```

5. マイグレーション Pod のログを確認し、トランザクションリカバリーがどのように実行されるかを確認します。リカバリーの終了まで待機します。

```
$ oc logs -f eap-app-migration-1-lvfdt
Finished Migration Check cycle, pausing for 30 seconds before resuming
...
Finished, recovery terminated successfully
Migration terminated with status 0 (T)
Releasing lock: (/opt/eap/standalone/partitioned_data/split-1)
Finished Migration Check cycle, pausing for 30 seconds before resuming
...
```

6. クラスターを元にスケールアップします。

```
$ oc scale --replicas=1 dc/eap-app
deploymentconfig "eap-app" scaled
```

7. ブラウザーを使用して再度 http://eap-app-eap-tx-demo.openshift.example.com/jboss-jta-crash-rec にアクセスします。

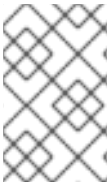
8. テーブルに両方のトランザクションのエントリーが含まれることに注意してください。以下の出力と似たものになります。

表7.1 例: データベーステーブルの内容

データベーステーブルの内容	
Key	Value
Mercedes	Benz updated via JMS.
Land	Rover updated via JMS.

上記のテーブルの内容は、2 つ目の XA トラザクションが終了する前にクラスターがスケールダウンしたものの、マイグレーション Pod がトランザクションリカバリーを実行し、トランザクションが正常に完了したことを示しています。

第8章 参考情報



注記

本セクションの内容は、このイメージの技術文書を参考にしました。開発の目的で便利なリファレンスとして提供され、製品ドキュメントの範囲外となるテスト用に提供されます。

8.1. 永続テンプレート

JBoss EAP およびデータベース Pod をデプロイする JBoss EAP データベーステンプレートは、一時的なものと永続的なものの両方があります。

永続テンプレートには、永続ボリュームクレームのプロビジョニングを行う環境変数が含まれます。これは、JBoss EAP for OpenShift デプロイメントのストレージボリュームとして使用される利用可能な永続ボリュームとバインドします。タイマースキーマ、ログ処理、またはデータ更新などの情報は、一時的なコンテナーメモリーではなく、ストレージボリュームに保存されます。この情報は、プロジェクトのアップグレード、デプロイメントのロールバック、予期せぬエラーなどの何らかの理由で Pod がダウンした場合に永続します。

デプロイメントの永続ストレージボリュームがないと、この情報はコンテナーメモリーのみに格納され、何らかの理由で Pod がダウンした場合は失われます。

たとえば、永続ストレージが基盤となる EE タイマーは Pod が再起動しても実行を継続します。再起動のプロセス中にタイマーによってトリガーされたイベントは、アプリケーションが再度稼働したとき実行されます。

逆に、EE タイマーがコンテナーメモリーで稼働している場合、Pod が再起動するとタイマーの状態は失われ、Pod が再度稼働したときに最初から開始します。

8.2. 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8.1 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-7/eap72-openshift**
JBOSS_IMAGE_RELEASE	イメージリリースラベル。 値: **dev**
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。

変数名	説明および値
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: **org.jboss.logmanager**、 **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**

8.3. 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。

表8.2 設定環境変数

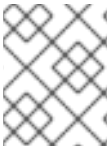
変数名	説明
AB_JOLOKIA_AUTH_OPENSHIFT	OpenShift TLS 通信のクライアント認証を切り替えます。このパラメーターの値は **true**、**false** または提供されるクライアントの証明書に含まれる必要がある相対識別名になります。デフォルトの CA 証明書は **/var/run/secrets/kubernetes.io/serviceaccount/ca.crt** に設定されます。 ● OpenShift TLS 通信のクライアント認証を無効にする場合は **false** に設定します。 ● デフォルトの CA 証明書とクライアントプリンシパルを使用して OpenShift TLS 通信のクライアント認証を有効にするには、**true** を設定します。 ● OpenShift TLS 通信のクライアント認証を有効にし、クライアントプリンシパルはオーバーライドするには、**cn=someSystem** などの相対識別名に設定します。提供されるクライアントの証明書にこの識別名が含まれている必要があります。
AB_JOLOKIA_CONFIG	設定した場合、Jolokia リファレンスドキュメント に説明があるように、この完全修飾ファイルパスを Jolokia JVM エージェントプロパティーに使用します。独自の Jolokia プロパティー設定ファイルを設定した場合、このドキュメントの残りの Jolokia 設定は無視されます。 設定しない場合、Jolokia リファレンスドキュメントに定義された設定を使用して、**/opt/jolokia/etc/jolokia.properties** が作成されます。 値の例: **/opt/jolokia/custom.properties**

変数名	説明
AB_JOLOKIA_DISCOVERY_ENABLED	Jolokia の検索を有効にします。 デフォルトは **false** です。
AB_JOLOKIA_HOST	バインド先のホストアドレス。 デフォルトは **0.0.0.0** です。 値の例: **127.0.0.1**
AB_JOLOKIA_HTTPS	HTTPS を使用したセキュアな通信を有効にします。 デフォルトでは、**AB_JOLOKIA_OPTS** に **serverCert** 設定の指定がないと、自己証明サーバー証明書が生成されます。 値の例: **true**
AB_JOLOKIA_ID	使用するエージェント ID。 デフォルト値はコンテナー ID の **$HOSTNAME** です。 値の例: **openjdk-app-1-xqlsj**
AB_JOLOKIA_OFF	**true** に設定すると、Jolokia のアクティベートを無効にし、空の値がエコーされます。 Jolokia はデフォルトで有効になります。
AB_JOLOKIA_OPTS	エージェント設定に追加されるその他のオプション。**key=value, key=value, …** の形式で指定する必要があります。 値の例: **backlog=20**
AB_JOLOKIA_PASSWORD	Basic 認証のパスワード。 デフォルトでは認証は無効になっています。 値の例: **mypassword**
AB_JOLOKIA_PASSWORD_RANDOM	**AB_JOLOKIA_PASSWORD** が無作為に生成されるべきかどうかを決定します。 パスワードを無作為に生成する場合は **true** に設定します。生成された値は **/opt/jolokia/etc/jolokia.pw** ファイルに保存されます。
AB_JOLOKIA_PORT	リッスンするポート。 デフォルトは **8778** です。 値の例: **5432**

変数名	説明
AB_JOLOKIA_USER	Basic 認証に使用するユーザーの名前。 デフォルトは **jolokia** です。 値の例: **myusername**
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、JMS 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 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**

変数名	説明
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**
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している場合、**JGROUPS_ENCRYPT_SECRET** 変数経由で指定される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	**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している場合、**JGROUPS_ENCRYPT_SECRET** 変数経由で指定されるシークレット内のキーストアファイルのディレクトリーパス。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: **/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**

変数名	説明
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_ENCRYPT_SECRET	**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、JGroups 通信のセキュア化に使用する JGroups キーストアファイルが含まれるシークレットの名前。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: **eap7-app-secret**
JGROUPS_PING_PROTOCOL	ノードの検索に使用する JGroups プロトコル。**openshift.DNS_PING** または **openshift.KUBE_PING** のいずれかになります。
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。



注記

上記に記載されていない、製品に影響するその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。

8.4. アプリケーションテンプレート

表8.3 アプリケーションテンプレート

変数名	説明
AUTO_DEPLOY_EXPLODED	展開形式のデプロイメントコンテンツが自動的にデプロイされるかどうかを制御します。 値の例: **false**

8.5. 公開されたポート

表8.4 公開されたポート

ポート番号	説明
8443	HTTPS
8778	Jolokia の監視

8.6. データソース

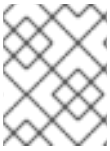
データソースは、環境変数の一部の値を元にして自動的に作成されます。

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。

8.6.1. 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。

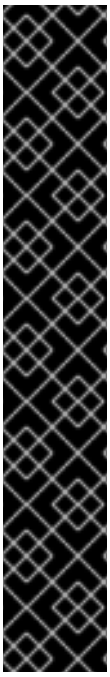
DATABASETYPE はデータソースのドライバーを決定します。

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。

8.6.1.1. データベースドライバー



重要

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、JDK 8 イメージストリームでは非推奨になりました。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が推奨されます。

以下の内部データソースは、JBoss EAP for OpenShift の JDK 11 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。

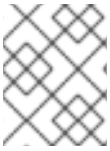
JBoss EAP で JDBC ドライバーを設定するための詳細は、『**設定ガイド**』の「JDBC ドライバー」を参照してください。

各 JDK 8 イメージには、デプロイされた MySQL、PostgreSQL、および MongoDB データベースの Java ドライバーが含まれています。データソースは MySQL および PostgreSQL データベースのみに生成されます。

重要

JDK 11 イメージストリームには、MySQL、PostgreSQL、および MongoDB データベースのドライバーは含まれず、データソースは生成されません。

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。

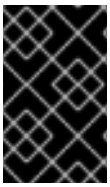


注記

MongoDB は SQL データベースでないため、MongoDB データベースには JNDI マッピングは作成されません。

8.6.1.2. 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、本セクションと「データソース」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**

変数名	説明
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**
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	非 XA データソースの Java Transaction API (JTA) オプションを定義します。XA データソースはすでにデフォルトで JTA 可能になっています。 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**

OpenShift でこのイメージを実行している場合、**POOLNAME_DATABASETYPE_SERVICE_HOST** および **POOLNAME_DATABASETYPE_SERVICE_PORT** 環境変数は OpenShift アプリケーションテンプレートのデータベースサービス定義から自動的に設定されます。 その他の環境変数は、各 Pod テンプレートのコンテナー定義の **env** エントリーとして直接テンプレートで設定されます。

8.6.1.3. 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。

8.6.1.3.1. 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。

8.6.1.3.2. 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。

8.7. クラスタリング

8.7.1. 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 **openshift.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-openshift.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

利用可能なアプリケーションテンプレートの1つを使用して、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、デフォルトの検索メカニズムは **openshift.DNS_PING** になります。

openshift.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 **openshift.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構成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。

8.7.1.1. 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。設定がないと、サーバーは単一ノードのクラスター (「1つのクラスター」) として動作します。例を以下に示します。

```
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。例を以下に示します。

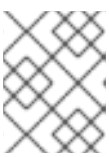
```
KUBERNETES_LABELS=application=APP_NAME
```

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 namespace 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に関する詳細は、「アプリケーションのデプロイメントに向けた OpenShift の準備」を参照してください。

8.7.1.2. 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 **openshift.DNS_PING** に設定します。

   ```
   JGROUPS_PING_PROTOCOL=openshift.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。設定のない場合、サーバーは単一ノードのクラスター (「1 つのクラスター」) のように動作します。

   ```
   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. **"true"** に設定された **service.alpha.kubernetes.io/tolerate-unready-endpoints** アノテーションをサービスに付ける必要があります。

   注記

   このアノテーションを省略すると、起動時に各ノードが独自の「1 つのクラスター」を構成することになり、他のノードは起動後まで検出されないため、起動後に他のノードのクラスターとマージすることになります。

   ```
   kind: Service
   apiVersion: v1
   spec:
      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。

8.7.2. 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-openshift.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構成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。

8.7.2.1. 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_SECRET** 環境変数を、JGroups の通信をセキュアにするために使用される JGroups キーストアファイルが含まれるシークレット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。例を以下に示します。

   ```
   JGROUPS_ENCRYPT_SECRET=eap7-app-secret
   ```

3. **JGROUPS_ENCRYPT_KEYSTORE_DIR** 環境変数を、**JGROUPS_ENCRYPT_SECRET** 変数を介して指定されるシークレット内にあるキーストアファイルのディレクトリーパス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。例を以下に示します。

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

4. **JGROUPS_ENCRYPT_KEYSTORE** 環境変数を、**JGROUPS_ENCRYPT_SECRET** 変数を介して指定されるシークレット内にある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。例を以下に示します。

```
JGROUPS_ENCRYPT_KEYSTORE=jgroups.jceks
```

5. **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。例を以下に示します。

```
JGROUPS_ENCRYPT_NAME=jgroups
```

6. **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。例を以下に示します。

```
JGROUPS_ENCRYPT_PASSWORD=mypassword
```

8.7.2.2. ASYM_ENCRYPT の設定

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、以下を行います。

1. **ASYM_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 **ASYM_ENCRYPT** に設定します。

```
JGROUPS_ENCRYPT_PROTOCOL=ASYM_ENCRYPT
```

2. **JGROUPS_CLUSTER_PASSWORD** 環境変数を、ノードの認証に使用されるパスワードに設定し、JGroups クラスターに参加できるようにする必要があります。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。例を以下に示します。

```
JGROUPS_CLUSTER_PASSWORD=mypassword
```

注記

ASYM_ENCRYPT を設定し、**SYM_ENCRYPT** に必要な環境変数も定義した場合、**SYM_ENCRYPT** 環境変数は無視され、警告が発生します。このような場合でも、クラスターの通信は **ASYM_ENCRYPT** を使用して暗号化されます。

8.8. ヘルスチェック

JBoss EAP for OpenShift イメージは、デフォルトで OpenShift に含まれる liveness および readiness プローブ を使用します。また、このイメージには 『**設定ガイド**』で説明されているように Eclipse MicroProfile Healthが含まれます。

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8.5 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
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
Eclipse MicroProfile Health [b]	N/A または **UP**	N/A または **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。

[b] **microprofile-health-smallrye** サブシステムが無効な場合、有効な状態は N/A のみ。

8.9. MESSAGING

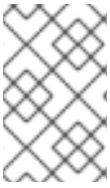
8.9.1. 外部 Red Hat AMQ ブローカーの設定

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。

OpenShift アプリケーション定義の例
以下の例はテンプレートを使用して、外部 Red Hat AMQ 7 ブローカーに接続する JBoss EAP アプリケーションを作成します。

例: eap72-amq-s2i アプリケーションテンプレート

```
oc new-app eap72-amq-s2i \
-p APPLICATION_NAME=eap72-mq \
-p MQ_USERNAME=MY_USERNAME \
-p MQ_PASSWORD=MY_PASSWORD
```



注記

以上の例では、JDK 11 イメージストリームは JDK 8 イメージストリームで使用される **eap72-amq-s2i** の代わりに、**eap72-openjdk11-amq-s2i** アプリケーションテンプレートを使用します。

重要

例で使用されるテンプレートは、必要なパラメーターに有効なデフォルト値を提供します。テンプレートを使用せず、独自のパラメーターを提供する場合は、**MQ_SERVICE_PREFIX_MAPPING** の名前が「-amq7=MQ」が追加された **APPLICATION_NAME** の名前と一致する必要があることに注意してください。

8.10. 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8.6 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。

8.11. 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**

変数名	説明
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**

8.12. 管理環境変数

表8.7 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**

8.13. 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。

8.13.1. 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-openshift.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。

8.13.1.1. 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。

8.13.2. 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** はカンマ区切りのリストです。例:
ARTIFACT_DIR=app1/target,app2/target,app3/target

8.13.3. 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必要なく、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る。

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、**http://10.0.0.1:8080/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
```

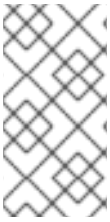
2. **eap** のビルド設定を **MAVEN_MIRROR_URL** 環境変数で更新します。

```
oc env bc/eap MAVEN_MIRROR_URL="http://10.0.0.1:8080/repository/internal/"
buildconfig "eap" updated
```

3. 設定を確認します。

```
oc env bc/eap --list
# buildconfigs eap
MAVEN_MIRROR_URL=http://10.0.0.1:8080/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。

8.13.4. 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 **openshift-launch.sh** 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。

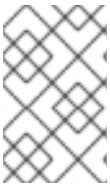
8.13.5. 環境変数

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8.8 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**
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**

変数名	説明
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: **http://10.0.0.1:8080/repository/internal/**
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。 値の例: **true**
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、JBoss EAP for OpenShift イメージに含まれる Maven および S2I スクリプトを使用する、「JBoss EAP for OpenShift イメージでの Java アプリケーションのビルドおよび実行」を参照してください。

8.14. SSO

このイメージには Red hat JBoss SSO 対応アプリケーションのサポートが含まれます。

注記

JBoss EAP for OpenShift イメージで OpenShift イメージの Red Hat JBoss SSO をデプロイする方法に関する詳細は、『Red Hat Single Sign-On for OpenShift』を参照してください。

表8.9 SSO 環境変数

変数名	説明
SSO_URL	SSO サーバーの URL。
SSO_REALM	デプロイされたアプリケーションの SSO レルム。
SSO_PUBLIC_KEY	SSO レルムの公開鍵。これは任意のフィールドですが、省略するとアプリケーションが中間者攻撃に脆弱になります。

変数名	説明
SSO_USERNAME	SSO REST API のアクセスに必要な SSO ユーザー。 値の例: **mySsoUser**
SSO_PASSWORD	**SSO_USERNAME** 変数によって定義された SSO ユーザーのパスワード。 値の例: **6fedmL3P**
SSO_SAML_KEYSTORE	SAML のキーストアの場所。デフォルトは **/etc/sso-saml-secret-volume/keystore.jks** です。
SSO_SAML_KEYSTORE_PASSWORD	SAML のキーストアパスワード。デフォルトは **mykeystorepass** です。
SSO_SAML_CERTIFICATE_NAME	SAML に使用するキー/証明書のエイリアス。デフォルトは **jboss** です。
SSO_BEARER_ONLY	SSO クライアントアクセスタイプ。（オプション） 値の例: **true**
SSO_CLIENT	アプリケーションへ戻る SSO リダイレクトのパス。デフォルトは **module-name** と一致します。
SSO_ENABLE_CORS	**true** の場合、SSO アプリケーションの CORS を有効にします。(任意)
SSO_SECRET	機密アクセスのための SSO クライアントシークレット。 値の例: **KZ1Qylq4**
SSO_DISABLE_SSL_CERTIFICATE_VALIDATION	**true** の場合、JBoss EAP と RH-SSO サーバー間の SSL/TLS 通信はセキュリティーで保護されません。たとえば、証明書の検証は **curl** で無効になります。デフォルトでは設定されません。 値の例: **true**

8.15. トランザクションリカバリー

クラスターがスケールダウンしたとき、トランザクションブランチがインダウトの状態になる可能性があります。このようなブランチを完了するためのテクノロジープレビューの自動化リカバリー Pod がありますが、リカバリーに失敗する可能性のあるネットワークスプリットなど、まれな状況が発生する場合があります。そのような場合、手動によるトランザクションリカバリーが必要になることがあります。

8.15.1. 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

親のネットワークエンドポイントはリカバリーコーディネーター IOR でエンコードされるため、子または親ノードのいずれかが新しい IP アドレスでリカバリーしたり、仮想化された IP アドレスを使用してアクセスされる目的である場合は、リカバリーが確実に動作しません。

- XTS トランザクション
  XTS はリカバリー目的ではクラスター化された状況で動作しません。詳細は「JBTM-2742」を参照してください。
- JBoss Remoting 上で伝搬されるトランザクション
- XATerminator 上で伝搬されるトランザクション
  EIS は、Java EE アプリケーションサーバーの単一インスタンスに接続することを目的とするため、これらのプロセスに対応する明確に定義された方法はありません。

8.15.2. 手動のトランザクションリカバリープロセス

次の手順の目的は、自動化リカバリーに失敗した場合にインダウト状態のブランチを見つけ、手作業で解決することです。

8.15.2.1. 注意事項

この手順は、単一の JVM 内で完全に自己完結したトランザクションを手動でリカバリーする方法のみを説明しています。この手順の説明は、別の JVM に伝搬された JTA トランザクションをリカバリーする方法ではありません。

重要

同じ IP アドレスと同じノード名を持つ同じ Pod の複数のインスタンスを OpenShift が起動する可能性があり、パーティションにより古い Pod が稼働するさまざまなネットワークパーティションのシナリオがあります。そのため、手動リカバリーの間に、オブジェクトストアの古いビューを持つ Pod に接続される可能性があります。これに該当すると思われる場合は、すべての JBoss EAP Pod をシャットダウンし、使用中のリソースマネージャーやオブジェクトストアが存在しないようにします。

XA トランザクションでリソースを登録する場合、各リソースタイプがリカバリーでサポートされることを確認するのはユーザーの責任となります。たとえば、PostgreSQL とMySQL はリカバリーで適切に動作することが知られています。 しかし、A-MQ や JDV リソースマネージャーはの場合は、特定の OpenShift リリースのドキュメントをチェックする必要があります。

デプロイメントは JDBC オブジェクトストア を使用する必要があります。

重要

トラザクションマネージャーは、ノード識別子が一意であることに依存します。XID の最大バイト長は XA 仕様によって設定され、変更できません。JBoss EAP for OpenShift イメージが XID に含めなければならないデータにより、ノード識別子には 23 バイトの空き領域があります。

OpenShift では以下をこの 23 バイトの制限に合わせるよう強制されます。

- すべてのノード名。 23 バイト未満の名前でも **-** (ダッシュ) は削除されます。
- 名前が 23 バイトを超える場合、名前の最初から 23 バイトの長さまでに省略されます。

しかし、これにより識別子の一意性に影響を与える可能性があります。たとえば、**aaa12345678901234567 8m0jwh** と **bbb12345678901234567 8m0jwh** は両方 **12345678901234567 8m0jwh** に省略され、名前の一意性が保たれなくなります。**this-pod-is-m0jwh** と **thispod-is-m0jwh** の場合でも、両方が **thispodism0jwh** に省略され、一意性が保たれなくなります。

上記の省略処理を念頭に置いて、設定するノード名が一意になるようにする必要があります。

8.15.2.2. 前提条件

OpenShift インスタンスは JDBC ストアと設定され、ストアテーブルは Pod 名に対応するテーブル接頭辞を使用してパーティションされることを前提とします。これは、JBoss EAP デプロイメントを使用する場合は常に自動である必要があります。これは、共有ボリューム上で split ディレクトリーを持つファイルストアを使用する 自動化リカバリーの例 とは異なります。稼働中の Pod で transaction サブシステムの設定を確認すると、JBoss EAP インスタンスが JDBC オブジェクトストアを使用していることが確認できます。

1. **/opt/eap/standalone/configuration/openshift-standalone.xml** 設定ファイルに transaction サブシステムの要素が含まれることを確認します。

```
<subsystem xmlns="urn:jboss:domain:transactions:3.0">
```

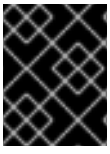
2. JDBC オブジェクトストアが使用されている場合、以下と似たエントリーが存在します。

```
<jdbc-store datasource-jndi-name="java:jboss/datasources/jdbcstore_postgresql"/>
```

注記

JNDI 名は、トランザクションログの格納に使用されるデータソースを識別します。

8.15.2.3. 手順



重要

以下の手順は、データソースのみに対する手動トランザクションリカバリーのプロセスについて説明します。

1. データベースベンダーのツールを使用して、インダウト状態のブランチの XID (トランザクショ

ンブランチ識別子) をリストします。失敗またはスケールダウンした **Pod で実行中であったすべてのデプロイメントが使用していたすべてのデータソース**の XID をリストする必要があります。使用しているデータベース製品のドキュメントを参照してください。

2. このような各 XID に対して、トランザクションを作成した Pod を特定し、その Pod が今も実行中であるかを確認します。

   a. 実行中である場合、ブランチはそのままにしておきます。

   b. Pod が実行していない場合、クラスターから削除されたと仮定し、ここで説明する手動の解決手順を適用する必要があります。失敗した Pod によって使用されたトランザクションログストレージに、対応するトランザクションログがあるかどうかを確認します。

      i. ログがある場合、ベンダーのツールを使用して XID を手動でコミットします。

      ii. ログがない場合、orphan ブランチであることを仮定し、ベンダーのツールを使用して XID をロールバックします。

これ以降の手順では、各ステップの実行方法を詳細に説明します。

8.15.2.3.1. インダウト状態のブランチの解決

最初に、デプロイメントが使用しているリソースをすべて探します。

この作業は、JBoss EAP 管理 CLI を使用して行うことが推奨されます。リソースは JBoss EAP の **standalone-openshift.xml** 設定ファイルに定義する必要がありますが、アプリケーションサーバー内で transaction サブシステムが利用できるようにする方法が他にあります。たとえば、デプロイメントのファイルを使用したり、ランタイムで管理 CLI を動的に使用したりして行うことができます。

1. 失敗した Pod のクラスターで JBoss EAP インスタンスを実行する Pod にてターミナルを開きます。対象の Pod がない場合は、その Pod に対してスケールアップします。

2. **/opt/eap/bin/add-user.sh** スクリプトを使用して管理ユーザーを作成します。

3. **/opt/eap/bin/jboss-cli.sh** スクリプトを使用して、管理 CLI にログインします。

4. サーバーに設定されたデータソースをリストします。これらにインダウト状態のトランザクションブランチが含まれる可能性があります。

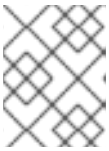
```
/subsystem=datasources:read-resource
{
    "outcome" => "success",
    "result" => {
     "data-source" => {
         "ExampleDS" => undefined,
         ...
     },
  ...
}
```

5. リストを表示したら、各データソースの接続 URL を見つけます。例を以下に示します。

```
/subsystem=datasources/data-source=ExampleDS:read-attribute(name=connection-url)
{
    "outcome" => "success",
```

```
    "result" => "jdbc:h2:mem:test;DB_CLOSE_DELAY=-1;DB_CLOSE_ON_EXIT=FALSE",
    "response-headers" => {"process-state" => "restart-required"}
}
```

6. 各データソースに接続し、インダウト状態のトランザクションブランチをすべて表示します。



注記

インダウト状態のブランチを格納するテーブル名は各データソースベンダーごとに異なります。

JBoss EAP には、各データベースをチェックするのに使用できるデフォルトの SQL クエリーツール (H2) があります。例を以下に示します。

```
java -cp /opt/eap/modules/system/layers/base/com/h2database/h2/main/h2-1.3.173.jar \
-url "jdbc:postgresql://localhost:5432/postgres" \
-user sa \
-password sa \
-sql "select gid from pg_prepared_xacts;"
```

この代わりに、リソースのネイティブツールを使用することもできます。たとえば、**sampledb** と呼ばれる PostGreSQL データソースでは、OpenShift クライアントツールを使用して Pod にリモートでログインし、インダウト状態のトランザクションテーブルにクエリーを実行することができます。

```
$ oc rsh postgresql-2-vwf9n # rsh to the named pod
sh-4.2$ psql sampledb
psql (9.5.7)
Type "help" for help.

sampledb=# select gid from pg_prepared_xacts;
131077_AAAAAAAAAAAAAP//rBEAB440GK1aJ72oAAAAGHAtanRhLWNyYXNoLXJlY0zLXAy
Y2N3_AAAAAAAAAAAAAP//rBEAB440GK1aJ72oAAAAGgAAAAEAAAAA
```

8.15.2.3.2. グローバルトランザクション ID およびノード識別子を各 XID から取得

インダウト状態のブランチの XID がすべて特定できたら、XID をトランザクションマネージャーのトランザクションテーブルに保存されたログと比較できる形式に変換します。

たとえば、この変換に以下の Bash スクリプトを使用することができます。**$PG_XID** が上記の select ステートメントからの XID を保持する場合、以下のように JBoss EAP トランザクション ID を取得することができます。

```
PG_XID="$1"
IFS='_' read -ra lines <<< "$PG_XID"
[[ "${lines[0]}" = 131077 ]] || exit 0; # this script only works for our own FORMAT ID
PG_TID=${lines[1]}

a=($(echo "$PG_TID"| base64 -d  | xxd -ps |tr -d '\n' | while read -N16 i ; do echo 0x$i ; done))
b=($(echo "$PG_TID"| base64 -d  | xxd -ps |tr -d '\n' | while read -N8 i ; do echo 0x$i ; done))
c=("${b[@]:4}") # put the last 3 32-bit hexadecimal numbers into array c
# the negative elements of c need special handling since printf below only works with positive
# hexadecimal numbers
for i in "${!c[@]}"; do
```

```
  arg=${c[$i]}
  # inspect the MSB to see if arg is negative - if so convert it from a 2's complement number
  [[ $(($arg>>31)) = 1 ]] && x=$(echo "obase=16; $(($arg - 0x100000000 ))" | bc) || x=$arg
  if [[ ${x:0:1} = \- ]] ; then # see if the first character is a minus sign
    neg[$i]="-";
    c[$i]=0x${x:1} # strip the minus sign and make it hex for use with printf below
  else
    neg[$i]=""
    c[$i]=$x
  fi
done
EAP_TID=$(printf %x:%x:${neg[0]}%x:${neg[1]}%x:${neg[2]}%x ${a[0]} ${a[1]} ${c[0]} ${c[1]} ${c[2]})
```

完了後、**$EAP_TID** 変数はこの XID を作成したトランザクションのグローバルトランザクション ID を保持します。トランザクションを開始した Pod のノード識別子は、以下の bash コマンドの出力によって提供されます。

```
echo "$PG_TID"| base64 -d | tail -c +29
```

注記

ノード識別子は、PostgreSQL グローバルトラザクション ID フィールドの 29 番目の文字から始まります。

- Pod が 実行している 場合は、トランザクションが実行中であるため、インダウト状態のブランチをそのままにしておきます。
- この Pod が実行していない場合は、トランザクションログの関連するトランザクションログストレージを検索する必要があります。ログストレージは、以下に従って命名される JDBC テーブルにあります。 **OS<node-identifier>jbosststxtable** パターン。
  - このようなテーブルがない場合は、他のトランザクションマネージャーによって所有されているため、ブランチをそのままにしておきます。このテーブルが含まれるデータソースの URL は、以下の transaction サブシステムで定義されます。
  - このようなテーブルがある場合、グローバルトランザクション ID と一致するエントリーを探します。
    - グローバルトランザクション ID と一致するエントリーがテーブルにある場合、以下の説明どおりにデータソースベンダーのツールを使用してインダウト状態のブランチをコミットする必要があります。
    - そのようなエントリーがない場合、ブランチは orphan であり、安全にロールバックすることができます。

以下は、インダウト状態の PostgreSQL ブランチをコミットする方法の例になります。

```
$ oc rsh postgresql-2-vwf9n
sh-4.2$ psql sampledb
psql (9.5.7)
Type "help" for help.
psql sampledb
commit prepared '131077_AAAAAAAAAAAAP//rBEAB440GK1aJ72oAAAAGHAtanRh
 ----
LWNyYXNoLXJlYy0zLXAyY2N3_AAAAAAAAAAAAP//rBEAB440GK1aJ72oAAAAGgAAAAEAAAAA';
```

重要

この手順をすべてのデータソースおよびインダウト状態のブランチに繰り返します。

8.15.2.3.3. リソースマネージャーに接触できるクラスターで稼働中すべての JBoss EAP インスタンスのノード識別子リストを取得

ノード識別子は、Pod 名と同じ名前になるように設定されます。**oc** コマンドを使用すると、使用中の Pod 名を取得できます。以下のコマンドを使用して、実行中の Pod をリストします。

```
$ oc get pods | grep Running
eap-manual-tx-recovery-app-4-26p4r   1/1       Running     0          23m
postgresql-2-vwf9n                   1/1       Running     0          41m
```

実行中の各 Pod に対し、Pod のログの出力を確認し、ノード名を取得します。たとえば、上記の出力にある最初の Pod の場合、以下のコマンドを使用します。

```
$ oc logs eap-manual-tx-recovery-app-4-26p4r | grep "jboss.node.name" | head -1
jboss.node.name = tx-recovery-app-4-26p4r
```

重要

前述の JBoss ノード名識別子 は、常に最大文字数である 23 文字になるよう省略されます。これは、先頭から文字を削除して最大長の 23 文字になるまで末尾の文字を確保します。

8.15.2.3.4. トランザクションログの検索

1. トランザクションログは JDBC が基盤のオブジェクトストアにあります。このストアの JNDI 名は、JBoss EAP 設定ファイルの **transaction** サブシステム定義に定義されています。

2. 設定ファイルを確認し、上記の JNDI 名に対応するデータソース定義を見つけます。

3. JNDI 名を使用して、接続 URL を取得します。

4. URL を使用してデータベースに接続し、関係するインダウト状態のトランザクションテーブルで **select** クエリーを実行します。
   データベースが実行している Pod が分かり、データベースの名前が分かる場合は、Pod に OpenShift リモートシェルを開き、直接データベースツールを使用した方が簡単であることがあります。

   たとえば、JDBC ストアが Pod **postgresql-2-vwf9n** で実行されている **sampledb** という PostgreSQL データベースによってホストされる場合、以下のコマンドを使用してトランザクションログを見つけることができます。

注記

以下のコマンドの **ostxrecoveryapp426p4rjbosststxtable** テーブル名は、ログストレージエントリーを保持する JDBC テーブル名のパターン にしたがっているため選択されました。ご使用の環境では、テーブル名は以下と似た形式になります。

- **os** 接頭辞で始まります。
- 中間部分は、上記の JBoss ノード名から適用されます。 「-」(ダッシュ) が存在する場合は削除される可能性があります。
- 最後に **jbosststxtable** 接頭辞が追加され、テーブルの最終名が作成されます。

```
$ oc rsh postgresql-2-vwf9n
sh-4.2$ psql sampledb
psql (9.5.7)
Type "help" for help.

sampledb=# select uidstring from ostxrecoveryapp426p4rjbosststxtable where
TYPENAME='StateManager/BasicAction/TwoPhaseCoordinator/AtomicAction'
;
             uidstring
 ------------------------------------
 0:ffff0a81009d:33789827:5a68b2bf:40
 (1 row)
```

8.15.2.3.5. 調整したインダウト状態のブランチのトランザクションログをクリーンアップ

警告

インダウト状態のブランチが残っていないことを確信できるまで、ログを削除しないでください。

指定のトラザクションのブランチがすべて完了し、A-MQ および JDV を含む潜在的なリソースマネージャーがすべてチェックされた後、トランザクションログを安全に削除できます。

以下のコマンドを実行します。 適切な **uidstring** を使用して削除するトラザクションログを指定します。

```
DELETE FROM ostxrecoveryapp426p4rjbosststxtable where uidstring = UIDSTRING
```

重要

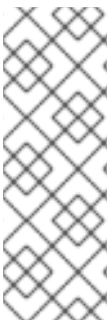
ログを削除しない場合、準備後に失敗し、現在は解決された完了済みのトランザクションはトランザクションログストレージから削除されません。この結果、不必要なストレージが使用され、今後の手作業の調整がより困難になります。

8.16. 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8.10 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.jboss.logmanager.ext
org.jgroups
org.mongodb
org.openshift.ping
org.postgresql
com.mysql
net.oauth.core



注記

以下のモジュールは、JDK 11 イメージには含まれていません。

- **org.mongodb**
- **org.postgresql**
- **com.mysql**

Revised on 2019-11-28 14:43:51 CET