Red Hat JBoss Enterprise Application Platform 7.3

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: 2023-09-02

Red Hat JBoss Enterprise Application Platform 7.3 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

Copyright © 2023 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at
http://creativecommons.org/licenses/by-sa/3.0/
. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

概要

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

目次

第1章 はじめに 4
1.1. RED HAT JBOSS ENTERPRISE APPLICATION PLATFORM (JBOSS EAP) とは 4
1.2. OPENSHIFT での JBOSS EAP の仕組み 4
1.3. 比較: JBOSS EAP および JBOSS EAP FOR OPENSHIFT 4
1.4. バージョンの互換性とサポート 5
1.5. テクノロジープレビューの機能 7
1.6. デプロイメントオプション 8

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 9
2.1. 要件 9
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 9
2.3. RED HAT コンテナーレジストリーへの認証の設定 10
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 11
2.5. JBOSS EAP S2I (SOURCE-TO-IMAGE) アプリケーションの OPENSHIFT へのデプロイ 13
2.6. デプロイメント後のタスク 15
2.7. JBOSS EAP FOR OPENSHIFT でのチェーンビルドのサポート 16

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 17
3.1. JBOSS EAP FOR OPENSHIFT の S2I プロセスの仕組み 17
3.2. 環境変数を使用した JBOSS EAP FOR OPENSHIFT の設定 18
3.3. ビルド拡張およびプロジェクトアーティファクト 27
3.4. OPENSHIFT での JBOSS EAP テンプレートの使用の結果 33
3.5. RED HAT JBOSS ENTERPRISE APPLICATION PLATFORM FOR OPENSHIFT IMAGES の SSO 設定 34
3.6. デフォルトデータソース 34
3.7. JBOSS EAP FOR OPENSHIFT イメージのデプロイメントに関する考慮事項 34

第4章 JBOSS EAP FOR OPENSHIFT の機能調整 37
4.1. カスタム JBOSS EAP サーバーのプロビジョニング 37
4.2. 利用可能な JBOSS EAP レイヤー 37
4.3. JBOSS EAP でのユーザー開発レイヤーのプロビジョニング 40

第5章 OPENSHIFT 4 の JBOSS EAP イメージストリームからの、EAP73 IMAGESTREAM へのアプリケーションの移行 45
5.1. EAP73 IMAGESTREAM の LIVENESS および READINESS プローブ設定の更新 45
5.2. デフォルトのデータソースが削除されました 46
5.3. OPENSHIFT 上の JBOSS EAP 7.1 を JBOSS EAP 7.3 にアップグレードするときに STANDALONE-OPENSHIFT.XML へ更新 46

第6章 トラブルシューティング 48
6.1. POD 再起動のトラブルシューティング 48
6.2. JBOSS EAP 管理 CLI を使用したトラブルシューティング 48

第7章 上級者向けチュートリアル 50
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 50

第8章 OPENSHIFT でのアプリケーションディプロイメントを自動化する EAP OPERATOR 55
8.1. WEB コンソールを使用した EAP OPERATOR のインストール 55
8.2. CLI を使用した EAP OPERATOR のインストール 57
8.3. EAP OPERATOR を使用した OPENSHIFT での JAVA アプリケーションデプロイ 58
8.4. EAP オペレーターを使用したアプリケーションのメトリクスの表示 67
8.5. WEB コンソールを使用した EAP OPERATOR のアンインストール 68
8.6. CLI を使用した EAP OPERATOR のアンインストール 68
8.7. 安全なトランザクションリカバリーの EAP OPERATOR 69

8.8. OPENSHIFT 上の EJB リモーティング 73

第9章 参考情報 76
9.1. 永続テンプレート 76
9.2. 情報環境変数 76
9.3. 設定環境変数 77
9.4. アプリケーションテンプレート 83
9.5. 公開されたポート 83
9.6. データソース 83
9.7. クラスタリング 87
9.8. ヘルスチェック 91
9.9. MESSAGING 92
9.10. セキュリティードメイン 92
9.11. HTTPS 環境変数 93
9.12. 管理環境変数 94
9.13. S2I 94
9.14. シングルサインオンイメージ 99
9.15. トランザクションリカバリー 101
9.16. 含まれる JBOSS モジュール 108
9.17. EAP OPERATOR: API 情報 108

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ーリング、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) 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 Jakarta 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、Web Profile および Full Platform 仕様の両方に対して Jakarta EE 8 と互換性のある実装で、Java EE 8 Full Platform および Web Profile 仕様の認定実装でもあります。

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。 EAP オペレーターは、OpenShift 4 におけるトランザクションリカバリーについて、テストおよびサポート対象の唯一のオプションです。EAP オペレーターを使用したトランザクションの回復の詳細は、EAP Operator for Safe Transaction Recovery を参照してください。
組み込みメッセージングブローカー	非推奨	OpenShift コンテナーでの組み込みメッセージングブローカーの使用は非推奨となりました。組み込みブローカーのサポートは今後のリリースで削除されます。 コンテナーが組み込みメッセージングブローカーを使用するよう設定され、リモートブローカーが設定されていない場合は、警告がログに記録されます。 コンテナー設定にメッセージング宛先が含まれていない場合は、**DISABLE_EMBEDDED_JMS_BROKER** 環境変数を **true** に設定して、埋め込みメッセージングブローカーを設定する機能を無効にします。

1.4. バージョンの互換性とサポート

JBoss EAP for OpenShift では JDK 8、JDK 11、および Eclipse OpenJ9 のイメージを利用できます。

各イメージの 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 ビルダーイメージには、S2I ビルド時に必要なツールを持つ完全な JBoss EAP サーバーが含まれます。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 for OpenShift 7.3 のイメージには、以下の変更が適用されています。

- デフォルトのドライバーおよびモジュールが削除されました。
- MySQL および PostgreSQL のテンプレートが削除されました。これらの機能は、カスタムレイヤーでプロビジョニングできます。

- Hawkular エージェントはこれらのイメージでアクティブではありません。設定されている場合は無視されます。
- デフォルトのデータソース **ExampleDS** は、コンテナーの起動時にデフォルトで追加されなくなりました。デフォルトのデータソースが必要な場合は、値が **true** (**ENABLE_GENERATE_DEFAULT_DATASOURCE=true**) の環境変数 **ENABLE_GENERATE_DEFAULT_DATASOURCE** を使用してこれを追加します。

注記

以下の検出メカニズムプロトコルは非推奨となり、他のプロトコルに置き換えられています。

- **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。**カスタマイズした standalone-openshift.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。

JDK 8 イメージ

- Red Hat Universal Base Image: 7
- テンプレート名の接頭辞: eap73-*
- ビルダーイメージ: https://access.redhat.com/containers/#/registry.access.redhat.com/jboss-eap-7/eap73-openjdk8-openshift-rhel7
- ランタイムイメージ: https://access.redhat.com/containers/#/registry.access.redhat.com/jboss-eap-7/eap73-openjdk8-runtime-openshift-rhel7

注記

JBoss EAP の JDK 8 イメージは、IBM Z および IBM Power Systems では提供されません。

JDK 11 イメージ

- Red Hat Universal Base Image: 8
- テンプレート名の接頭辞: eap73-openjdk11-*
- ビルダーイメージ: https://access.redhat.com/containers/#/registry.access.redhat.com/jboss-eap-7/eap73-openjdk11-openshift-rhel8
- ランタイムイメージ: https://access.redhat.com/containers/#/registry.access.redhat.com/jboss-eap-7/eap73-openjdk11-runtime-openshift-rhel8

Eclipse OpenJ9 イメージ

- Red Hat Universal Base Image: 8

- テンプレート名の接頭辞: eap73-*
- ビルダーイメージ: https://access.redhat.com/containers/#/registry.access.redhat.com/jboss-eap-7/eap73-openj9-11-openshift-rhel8:
- ランタイムイメージ: https://access.redhat.com/containers/#/registry.access.redhat.com/jboss-eap-7/eap73-openj9-11-runtime-openshift-rhel8:

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 and Atomic Platform Tested Integrations 参照してください。

その他のリソース

JBoss EAP for OpenShift の機能調整

1.4.1. 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。

1.4.2. IBM Z および IBM Power Systems のサポート

libartemis-native の s390x および ppc64le 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。

1.4.3. OpenShift での JBoss EAP 7.1 から JBoss EAP 7.3 へのアップグレード

OpenShift において JBoss EAP 7.1 でインストールされたファイル **standalone-openshift.xml** は、JBoss EAP 7.3 以降と互換性がありません。OpenShift 用の JBoss EAP 7.3 以降のコンテナーを起動するには、JBoss EAP 7.1 でインストールされた **standalone-openshift.xml** ファイルを変更する必要があります。

その他のリソース

OpenShift 上の JBoss EAP 7.1 を JBoss EAP 7.3 にアップグレードするときに **standalone-openshift.xml** へ更新

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 テクノロジープレビュー機能のサポート範囲 を参照してください。

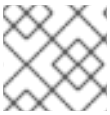
クラスターがスケールダウンしたとき、トランザクションブランチがインダウトの状態になる可能性があります。JBoss EAP for OpenShift イメージには、これらのブランチを完了できる 自動トランザクションリカバリー機能 があります。現時点では、この実装の自動化トランザクションリカバリーはテクノロジープレビューとしてのみ提供されます。

アプリケーション Pod のスケールダウンでの自動化トランザクションリカバリーの実証に提供される **eap73-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 **eap73-openjdk11-tx-recovery-s2i** です。

1.6. デプロイメントオプション

以下のオプションのいずれかを使用して、JBoss EAP Java アプリケーションを OpenShift にデプロイできます。

- JBoss EAP for OpenShift テンプレート。詳細は、Build and Run a Java Application on the JBoss EAP CD for OpenShift Image を参照してください。
- EAP オペレーターは JBoss EAP 固有のコントローラーです。これは、OpenShift ユーザーの代わりに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、管理を行うために OpenShift API を拡張します。詳細は、OpenShift でのアプリケーションディプロイメントを自動化する EAP Operator を参照してください。



注記

EAP オペレーターは、OpenShift 4 以降のバージョンでのみサポートされます。

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 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 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、**eap73-https-s2i** テンプレート (JDK 8 用) または **eap73-openjdk11-https-s2i** テンプレート (JDK 11 用) を使用している場合は、キーストアとシークレットを作成する必要があります。

   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 create secret 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 create secret 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

JDK の最新の JBoss EAP for OpenShift イメージストリームおよびテンプレートを OpenShift プロジェクトの namespace にインポートする必要があります。

注記

カスタマーポータルの認証情報を使用して Red Hat Container Registry にログインし、JBoss EAP イメージストリームおよびテンプレートをインポートします。詳細は、Red Hat コンテナーレジストリーの認証 を参照してください。

JDK 8 の import コマンド

```
for resource in \
  eap73-amq-persistent-s2i.json \
  eap73-amq-s2i.json \
  eap73-basic-s2i.json \
  eap73-https-s2i.json \
  eap73-image-stream.json \
  eap73-sso-s2i.json \
  eap73-starter-s2i.json \
  eap73-third-party-db-s2i.json \
  eap73-tx-recovery-s2i.json
do
  oc replace --force -f \
https://raw.githubusercontent.com/jboss-container-images/jboss-eap-7-openshift-
image/eap73/templates/${resource}
done
```

このコマンドは以下のイメージストリームおよびテンプレートをインポートします。

- JDK 8 ビルダーイメージストリーム: **jboss-eap73-openshift**
- JDK 8 ランタイムイメージストリーム: **jboss-eap73-runtime-openshift**
- コマンドで指定したすべてのテンプレート

JDK 11 の import コマンド

```
for resource in \
  eap73-openjdk11-amq-persistent-s2i.json \
```

```
  eap73-openjdk11-amq-s2i.json \
  eap73-openjdk11-basic-s2i.json \
  eap73-openjdk11-https-s2i.json \
  eap73-openjdk11-image-stream.json \
  eap73-openjdk11-sso-s2i.json \
  eap73-openjdk11-starter-s2i.json \
  eap73-openjdk11-third-party-db-s2i.json \
  eap73-openjdk11-tx-recovery-s2i.json
do
  oc replace --force -f \
https://raw.githubusercontent.com/jboss-container-images/jboss-eap-7-openshift-
image/eap73/templates/${resource}
done
```

このコマンドは以下のイメージストリームおよびテンプレートをインポートします。

- JDK 11 ビルダーイメージストリーム: **jboss-eap73-openjdk11-openshift**
- JDK 11 ランタイムイメージストリーム: **jboss-eap73-openjdk11-runtime-openshift**
- コマンドで指定したすべてのテンプレート

IBM Z および IBM Power Systems における Eclipse OpenJ9 の import コマンド

```
oc replace --force -f \
https://raw.githubusercontent.com/jboss-container-images/jboss-eap-openshift-
templates/eap73/eap73-openj9-image-stream.json

for resource in \
  eap73-amq-persistent-s2i.json \
  eap73-amq-s2i.json \
  eap73-basic-s2i.json \
  eap73-https-s2i.json \
  eap73-sso-s2i.json \
  eap73-third-party-db-s2i.json \
  eap73-tx-recovery-s2i.json
do
  oc replace --force -f \
https://raw.githubusercontent.com/jboss-container-images/jboss-eap-openshift-
templates/eap73/templates/${resource}
done
```

このコマンドは以下のイメージストリームおよびテンプレートをインポートします。

- Eclipese OpenJ9 ビルダーイメージ: **jboss-eap-7-OpenJ9-11-openshift:**
- Eclipse OpenJ9 ランタイムイメージストリーム: **jboss-eap-7-OpenJ9-11-runtime-openshift**
- コマンドで指定したすべてのテンプレート

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。以下に例を示します。

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

cluster-samples-operator を使用する場合は、クラスターサンプルオペレーターの設定についての OpenShift ドキュメントを参照してください。クラスターサンプルオペレーターの設定の詳細は、サンプルオペレーターの設定 を参照してください。

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

イメージおよびテンプレートのインポート後に、アプリケーションを OpenShift にデプロイできます。

OpenJDK 8 および Eclipse OpenJ9 のテンプレート名は接頭辞 **eap73-*** を使用します (例: **eap73-https-s2i**)。OpenJDK 11 テンプレート名は接頭辞 **eap73-openjdk11-*** を使用します (例: **eap73-openjdk11-https-s2i**)。

要件

オプション: 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。

手順

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。S2I ビルド用に提供される JBoss EAP for OpenShift テンプレートのいずれかを使用できます。トリムされたサーバーのプロビジョニングも選択できます。
   たとえば、JDK 8 ビルダーイメージを使用して **kitchensink** クイックスタートをディプロイするには、アプリケーションのデプロイメントに向けた OpenShift の準備 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 **eap73-basic-s2i** テンプレートを使用します。このクイックスタートはトリム機能に対応していません。

```
oc new-app --template=eap73-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=7.3.x-openshift \ (4)
 -p CONTEXT_DIR=kitchensink (5)
```

(1) 使用するテンプレート。**eap73** 接頭辞は JDK 8 テンプレートを指定します。

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

IBM Z および IBM Power Systems の Eclipse OpenJ9 ビルダーイメージに、このコマンドの変更バージョンを使用します。コマンドで以下のイメージ名パラメーターを追加します。JDK 環境では、これらのパラメーターにデフォルト値が使用されます。

- EAP_IMAGE_NAME=jboss-eap-7-openj9-11-openshift \
- EAP_RUNTIME_IMAGE_NAME=jboss-eap-7-openj9-11-runtime-openshift \

別の例として、JDK 11 ランタイムイメージを使用し、JBoss EAP をトリムして **helloworld-html5** クイックスタートをデプロイするには **jaxrs-server** レイヤーのみを含め、以下のコマンドを入力します。このコマンドは、GitHub の **helloworld-html5** ソースコードとともに アプリケーションのデプロイメントに向けた OpenShift の準備 で作成した、**eap-demo** プロジェクトで **eap73-openjdk11-basic-s2i** テンプレートを使用します。

```
oc new-app --template=eap73-openjdk11-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=7.3.x-openshift \ (4)
 -p GALLEON_PROVISION_LAYERS=jaxrs-server \ (5)
 -p CONTEXT_DIR=helloworld-html5 (6)
```

1. 使用するテンプレート。**eap73-openjdk11** 接頭辞は、JDK 11 テンプレートを指定します。

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. **jaxrs-server** レイヤーのみを持つトリムされたサーバーをプロビジョニングします。

6. 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

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、**eap73-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。

注記

テンプレートで AMQ が使用される場合は、**AMQ_IMAGE_NAME** パラメーターに適切な値を含める必要があります。

テンプレートが SSO を使用する場合は、適切な値を指定して **SSO_IMAGE_NAME** パラメーターを含める必要があります。

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
```

関連情報

JBoss EAP for OpenShift の機能調整

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ある場合があります。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
```

2.7. JBOSS EAP FOR OPENSHIFT でのチェーンビルドのサポート

JBoss EAP for OpenShift は OpenShift でのチェーンビルドをサポートします。

JBoss EAP for OpenShift テンプレートはチェーンビルドを採用しています。これらのテンプレートを使用する場合は、ビルドの結果は以下のようになります。

- **[application name]-build-artifacts** という名前の中間イメージ
- 最終的なイメージ **[アプリケーション名]**

チェーンビルドの詳細は、OpenShift ドキュメントを参照してください。

関連情報

OpenShift Chained build documentation

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

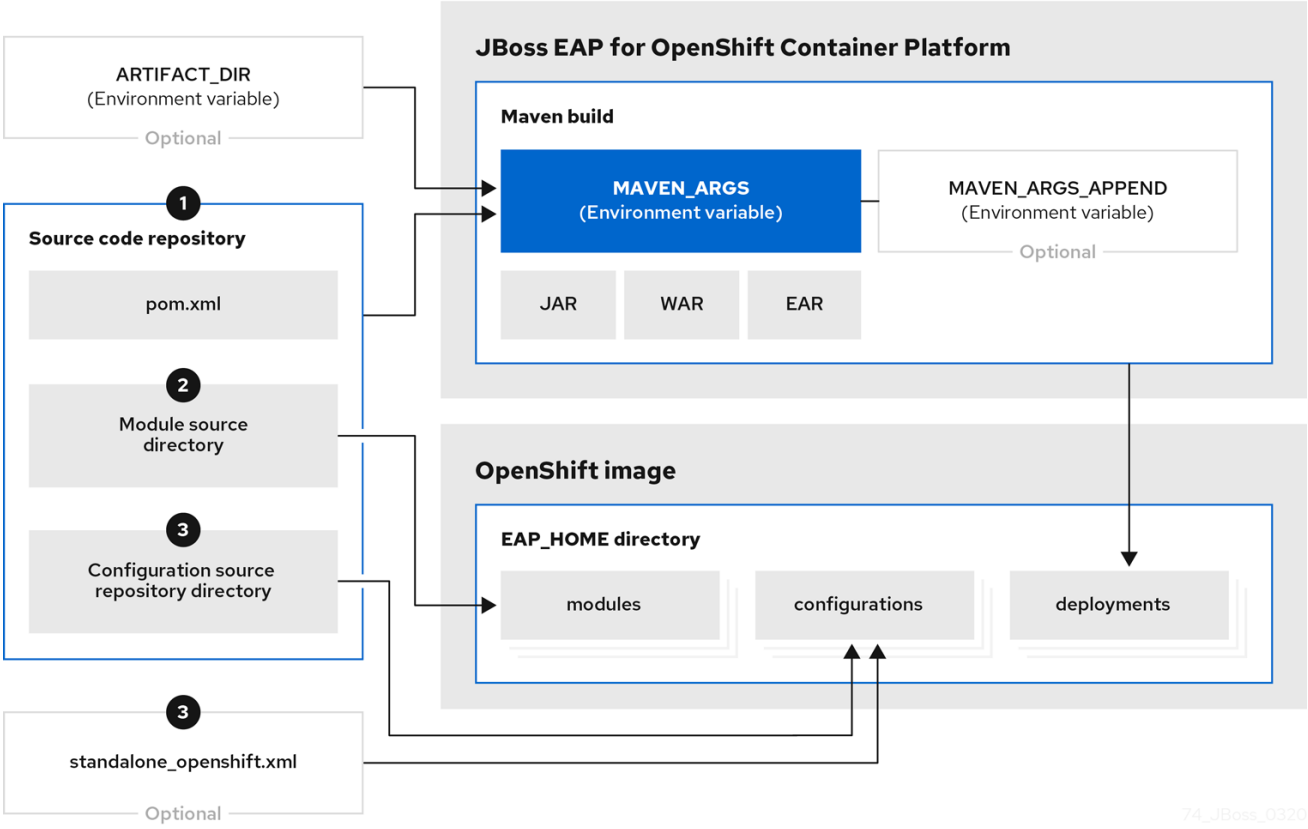
コンテナーが再起動または終了すると、実行中のコンテナーで変更された設定内容はすべて失われます。

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

JBoss EAP の S2I プロセスを示すフローチャート:



1. **pom.xml** ファイルがソースコードリポジトリーにある場合、S2I ビルダーイメージは Maven ビルドプロセスを開始します。Maven ビルドは **$MAVEN_ARGS** の内容を使用します。**pom.xml** ファイルがソースコードリポジトリーにない場合、S2I ビルダーイメージはバイナリータイプのビルドを開始します。

カスタム Maven 引数またはオプションを追加するには、**$MAVEN_ARGS_APPEND** を使用します。**$MAVEN_ARGS_APPEND** 変数は、**$MAVEN_ARGS** に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。

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。

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. **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。

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は、ファイル名を **standalone-openshift.xml** にする必要があります。

関連情報

- バイナリータイプのビルドについての詳細は、OpenShift 4.2 ドキュメントの バイナリー (ローカル) ソースを参照してください。
- S2I プロセスを指示してカスタム Maven アーティファクトリー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ーポジトリーミラー 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=eap73-basic-s2i \
 -p IMAGE_STREAM_NAMESPACE=eap-demo \
 -p SOURCE_REPOSITORY_URL=https://github.com/jboss-developer/jboss-eap-quickstarts \
 -p SOURCE_REPOSITORY_REF=7.3.x-openshift \
 -p CONTEXT_DIR=kitchensink \
 -e ADMIN_USERNAME=myspecialuser \
 -e ADMIN_PASSWORD=myspecialp@ssw0rd
```

注記

この例では、JDK 8 テンプレートを使用します。JDK 11 の場合は、**eap73-openjdk11-basic-s2i** テンプレートを使用します。

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。

3.2.1. JVM のメモリー設定

OpenShift EAP イメージには、現在の環境に基づいてデフォルトの JVM メモリー設定を自動的に計算するメカニズムがあります。ただし、環境変数を使用して JVM メモリーを設定することも可能です。

3.2.1.1. JVM のデフォルトメモリー設定

現在のコンテナーに対してメモリー制限が定義されており、この制限が利用可能なメモリーの合計よりも小さい場合、デフォルトの JVM メモリー設定は自動的に算出されます。それ以外の場合、デフォルトの JVM メモリー設定は、イメージのベースサーバーとして使用される EAP バージョンの **standalone.conf** ファイルで定義されます。

コンテナーのメモリー制限は、**/sys/fs/cgroup/memory/memory.limit_in_bytes** ファイルから取得されます。利用可能なメモリーの合計は、**/proc/meminfo** コマンドで取得されます。

メモリー設定が自動的に算出されると、以下の式が使用されます。

- 最大ヒープサイズ (-Xmx): ユーザーメモリーの 50%
- 初期ヒープサイズ (-Xms): 計算済み最大ヒープサイズの 25%。

たとえば、定義したメモリー制限が 1 GB で、この制限が **/proc/meminfo** に示されている利用可能なメモリー合計よりも低い場合、そのメモリー設定は -Xms128m -Xmx512 になります。

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 (つまり、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)。

- **JAVA_MAX_MEM_RATIO**
- **JAVA_INITIAL_MEM_RATIO**
- **JAVA_MAX_INITIAL_MEM**

以下の 2 つの環境変数の値を 0 に設定すると、メモリーの自動計算を無効にできます。

- **JAVA_INITIAL_MEM_RATIO**
- **JAVA_MAX_MEM_RATIO**

3.2.1.2. 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:+UseParallelOldGC -XX:MinHeapFreeRatio=10 -XX:MaxHeapFreeRatio=20 -XX:GCTimeRatio=4 -XX:AdaptiveSizePolicyWeight=90 -XX:+ExitOnOutOfMemoryError

Java 8 のガベージコレクションのロギング設定 (非モジュール JVM)

-verbose:gc -Xloggc:/opt/eap/standalone/log/gc.log -XX:+PrintGCDetails -XX:+PrintGCDateStamps -XX:+UseGCLogFileRotation -XX:NumberOfGCLogFiles=5 -XX:GCLogFileSize=3M -XX:-TraceClassUnloading

Java 11 (modular JVM) のガベージコレクションのロギング設定

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

3.2.1.3. デフォルト設定のリソース制限

これが設定されている場合には、追加のデフォルト設定がイメージに含まれます。

-XX:ParallelGCThreads={core-limit} -Djava.util.concurrent.ForkJoinPool.common.parallelism={core-limit} -XX:CICompilerCount=2

{core-limit} の値は、**JAVA_CORE_LIMIT** 環境変数を使用するか、コンテナーによる CPU コア制限で定義されます。

CICompilerCount の値は常に 2 に固定されます。

3.2.1.4. 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3.1 JVM 環境変数

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

変数名	例	デフォルト値	JVM の設定	説明
				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=org.jboss.logmanager,jdk.nashorn..api,com.sun.crypto.provider ● -Djava.awt.headless=true さらに、自動メモリー計算が有効になっていない場合、初期 Java メモリー (-Xms) および最大 Java メモリー (-Xmx) は定義されません。 **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上書きします。
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。

変数名	例	デフォルト値	JVM の設定	説明
JAVA_MAX_MEM_RATIO	50	50	-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	25	-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。

変数名	例	デフォルト値	JVM の設定	説明
JAVA_MAX_INITIAL_MEM	4096	4096	-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、初期メモリーヒープの最大サイズの算出に使用されます。この値はメガバイト (MB) で示されます。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、**-Xms** の値は、その変数で指定された値に設定されます。デフォルト値 4096 では、初期ヒープの最大値が 4096 MB を超えることはありません。

変数名	例	デフォルト値	JVM の設定	説明
JAVA_DIAGNOSTICS	true	false (無効)	この設定は、コンテナーが使用する JDK によって異なります。 ● OpenJDK 8: -XX:NativeMemoryTracking=summary -XX:+PrintGC -XX:+PrintGCDateStamps -XX:+PrintGCTimeStamps -XX:+UnlockDiagnosticVMOptions ● OpenJDK 11: -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** が既に **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。

変数名	例	デフォルト値	JVM の設定	説明
JAVA_CORE_LIMIT		未定義	-XX:parallelGCThreads -Djava.util.concurrent.ForkJoinPool.common.parallelism -XX:CICompilerCount	コア数におけるユーザー定義の制限。コンテナーが制限の制約を報告する場合、JVM 設定の値はコンテナーのコア制限に制限されます。-XXCICompilerCount の値は、常に 2 です。デフォルトでは、この変数は未定義です。この場合、制限がコンテナーに定義されていなければ、JVM 設定は設定されません。
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。
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。

変数名	例	デフォルト値	JVM の設定	説明
GC_MAX_METASPACE_SIZE	100	256	-XX:MaxMetaspaceSize	最大メタスペースサイズ。
GC_CONTAINER_OPTIONS	-XX:+UserG1GC	-XX:-UseParallelOldGC	-XX:-UseParallelOldGC	使用する Java ガベージコレクションを指定します。この変数の値は、必要なガベージコレクションを指定するための JRE コマンドラインオプションである必要があります。JRE コマンドはデフォルトをオーバーライドします。

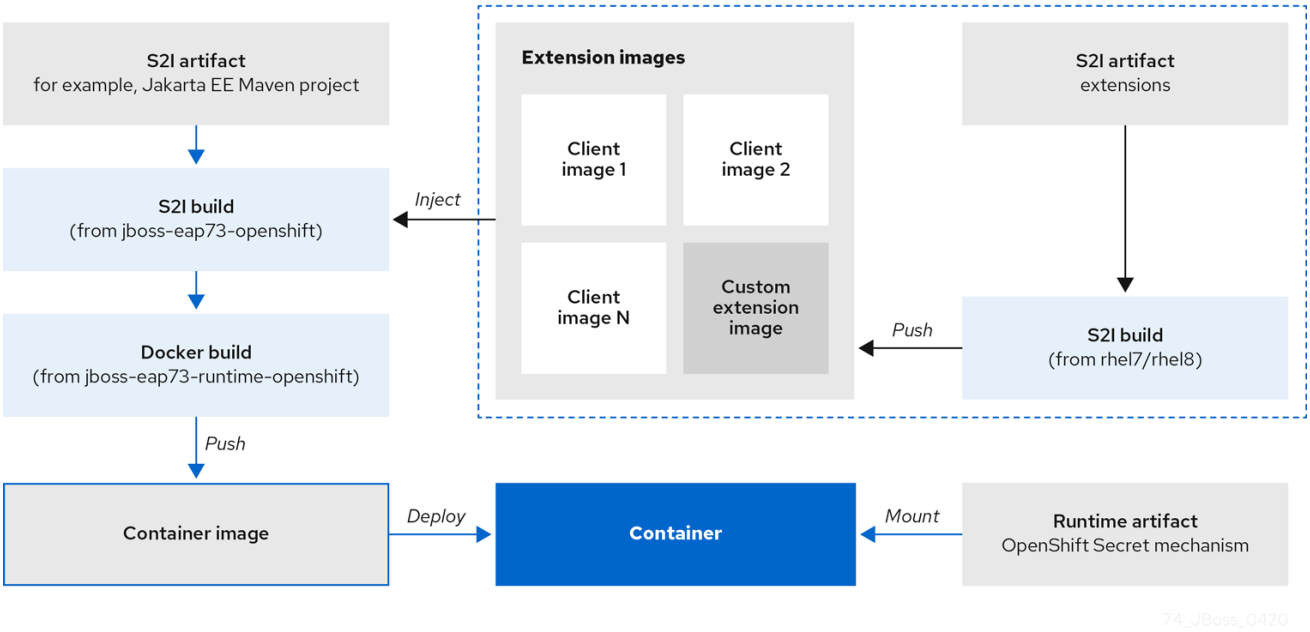
以下の環境変数が非推奨になりました。

- **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。

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、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お勧めします。

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。

関連情報

JBoss EAP for OpenShift の機能調整

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ー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ーポジトリーミラー 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します。**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。

       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。

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。これらのドライバーをモジュールとしてインストールできます。モジュール、ドライバー、および汎用デプロイメント の説明を参照してください。これらの JDBC ドライバーは、JBoss EAP アプリケーションのデータベースベンダーから取得できます。

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=postgresql
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

データソースには、以下の 2 つのタイプがあります。

1. 内部データソース。これらデータソースは、OpenShift 上で実行されますが、Red Hat レポジトリーからはデフォルトで利用できません。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。
2. 外部データソース。これらのデータソースは OpenShift 上では動作しません。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

```
DB_SERVICE_PREFIX_MAPPING=PostgresXA-POSTGRES=DS1
DS1_JNDI=java:jboss/datasources/pgds
DS1_DRIVER=postgresql-42.2.5.jar
DS1_USERNAME=postgres
DS1_PASSWORD=postgres
```

```
DS1_MAX_POOL_SIZE=20
DS1_MIN_POOL_SIZE=20
DS1_CONNECTION_CHECKER=org.jboss.jca.adapters.jdbc.extensions.postgres.PostgreSQLValidConnectionChecker
DS1_EXCEPTION_SORTER=org.jboss.jca.adapters.jdbc.extensions.postgres.PostgreSQLExceptionSorter
```

DB_SERVICE_PREFIX_MAPPING 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。接続プール固有のプロパティーはデータソースへの接続を生成します。データベースドライバー固有のプロパティーはデータソースのドライバーを決定し、ドライバー S2I アーティファクトとして設定されます。

上記の例では、**DS1** はデータソース接頭辞です。**CONNECTION_CHECKER** はデータベースの接続の検証に使用される接続チェッカークラスを指定し、**EXCEPTION_SORTER** は致命的なデータベース接続例外検出に使用される例外ソータークラスを指定します。

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3.2 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。

属性	説明
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. OPENSHIFT での JBOSS EAP テンプレートの使用の結果

JBoss EAP テンプレートを使用してアプリケーションをコンパイルする場合は、以下のイメージが生成されることがあります。

最終イメージ **[application name]** の作成前に **[application name]-build-artifacts** という名前の中間イメージが生成されることがあります。

[application name]-build-artifacts イメージは、アプリケーションがデプロイされた後に削除できます。

3.5. RED HAT JBOSS ENTERPRISE APPLICATION PLATFORM FOR OPENSHIFT IMAGES の SSO 設定

Red Hat JBoss Enterprise Application Platform for OpenShift イメージでは、SSO はレガシー **security** サブシステムを使用するように設定されます。

これらのイメージでは、環境変数 **SSO_FORCE_LEGACY_SECURITY** が **true** に設定されます。

SSO セキュリティーに **elytron** サブシステムを使用する場合は、**SSO_FORCE_LEGACY_SECURITY** 環境変数の値を **false** に更新します。

3.6. デフォルトデータソース

JBoss EAP 7.3 では、デフォルトのデータソース **ExampleDS** が削除されます。

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、デフォルト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、**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
```

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

3.7.1. 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。

マルチ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。

警告

EAP オペレーターを使用しているときに **SPLIT_DATA** などの環境変数を使用すると、一貫性の問題が発生する可能性があります。EAP オペレーターを使用して、OpenShift 4 以降のバージョンでトランザクション検出を管理する必要があります。

これは、トランザクションリカバリーテンプレート (JDK 8 の場合は **eap73-tx-recovery-s2i**、JDK 11 の場合は **eap73-openjdk11-tx-recovery-s2i**) のデフォルト設定となりました。

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。

3.7.2. 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、予期せず終了したトランザクションが残る場合があります。

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、JBoss EAP トランザクションリカバリーテンプレート (JDK 8 の場合は **eap73-tx-recovery-s2i**、JDK 11 の場合は **eap73-openjdk11-tx-recovery-s2i**) はトランザクションの移行を管理する移行 Pod を含む次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 の機能調整

JBoss EAP を含むイメージを構築しているとき、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。

S2I イメージに含まれるデフォルトの JBoss EAP サーバーには、完全なサーバーおよびすべての機能が含まれます。プロビジョニングしたサーバーに含まれる機能を調整したいこともあります。たとえば、プロビジョニングされたサーバーのセキュリティーの露出を低減させたり、マイクロサービスのコンテナーにより適切になるように、メモリーフットプリントを低減させる必要がある場合があります。

4.1. カスタム JBOSS EAP サーバーのプロビジョニング

調整した機能を備えたカスタムサーバーをプロビジョニングするに **は、S2I ビルドフェーズで Galleon_provision_LAYERS** 環境変数を渡します。

環境変数の値は、サーバーのビルドにプロビジョニングするコンマ区切りのレイヤーの一覧です。

たとえば、環境変数を **GALLEON_PROVISION_LAYERS=jaxrs-server** として指定すると、JBoss EAP サーバーは以下の機能を備えた状態でプロビジョニングされます。

- サーブレットコンテナー
- データソースを設定する機能
- **jaxrs**、**weld**、**jpa** サブシステム
- Red Hat SSO 統合

4.2. 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ように 6 つの層を提供しています。

3 つの層は、コア機能を提供するベースレイヤーです。デコレーターは、ベースレイヤーを強化するデコレーター層です。

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

4.2.1. 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。

このレイヤーには MicroProfile 機能が含まれません。

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**

このレイヤーは、コンテナーに Infinispan ベースのセカンドレベルのエンティティーキャッシングをローカルに追加します。

以下の MicroProfile 機能は、このレイヤーに含まれています。

- MicroProfile REST クライアント

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。

- MicroProfile Health
- MicroProfile Metrics
- MicroProfile Config
- MicroProfile OpenTracing

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

4.2.2. 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とで、追加機能を利用できます。

sso
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 Red Hat Single Sign-On 統合を追加します。

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。

- MicroProfile Health
- MicroProfile Metrics
- MicroProfile Config
- MicroProfile OpenTracing

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ー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。

4.3. JBOSS EAP でのユーザー開発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、Building Custom Layers for JBoss EAP 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. 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。
   詳細は、Custom Provisioning Files for JBoss EAP を参照してください。
4. 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、 Building an Application Provisioned with User-developed Layers を参照してください。

4.3.1. JBoss EAP のカスタムレイヤーの構築

カスタムレイヤー機能パックを Maven プロジェクトとして作成します。

1. カスタムレイヤーは、少なくともベースレイヤーに依存します。カスタムレイヤーに必要な機能を提供するベースレイヤーを選択します。
2. Maven プロジェクト内で、**src/main/resources** ディレクトリーにレイヤーコンテンツを作成します。
   たとえば、PostgreSQL および PostgreSQL データソースのサポートをプロビジョニングするレイヤーを作成するには、**src/main/resources** ディレクトリーに **layers/standalone** サブディレクトリーを作成します。**standalone** サブディレクトリーには以下の内容が含まれます。

- **postgresql-driver**
  このディレクトリーには、以下の内容が含まれる **layer-spec.xml** ファイルが含まれます。

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

- **postgresql-datasource**
  このディレクトリーには、以下の内容が含まれる **layer-spec.xml** ファイルが含まれます。

```
<?xml version="1.0" ?>
<layer-spec xmlns="urn:jboss:galleon:layer-spec:1.0" name="postgresql-datasource">
   <dependencies>
      <layer name="postgresql-driver"/>
   </dependencies>
   <feature spec="subsystem.datasources.data-source">
      <param name="use-ccm" value="true"/>
      <param name="data-source" value="PostgreSQLDS"/>
      <param name="enabled" value="true"/>
      <param name="use-java-context" value="true"/>
      <param name="jndi-name"
value="java:jboss/datasources/${env.POSTGRESQL_DATASOURCE,env.OPENSHIFT_P
OSTGRESQL_DATASOURCE:PostgreSQLDS}"/>
      <param name="connection-url"
value="jdbc:postgresql://${env.POSTGRESQL_SERVICE_HOST,\
env.OPENSHIFT_POSTGRESQL_DB_HOST}:${env.POSTGRESQL_SERVICE_PORT,\

env.OPENSHIFT_POSTGRESQL_DB_PORT}/${env.POSTGRESQL_DATABASE,
env.OPENSHIFT_POSTGRESQL_DB_NAME}"/>
      <param name="driver-name" value="postgresql"/>
      <param name="user-name" value="${env.POSTGRESQL_USER,
env.OPENSHIFT_POSTGRESQL_DB_USERNAME}"/>
      <param name="password" value="${env.POSTGRESQL_PASSWORD,
env.OPENSHIFT_POSTGRESQL_DB_PASSWORD}"/>
      <param name="check-valid-connection-sql" value="SELECT 1"/>
      <param name="background-validation" value="true"/>
      <param name="background-validation-millis" value="60000"/>
      <param name="flush-strategy" value="IdleConnections"/>
      <param name="statistics-enabled" value="${wildfly.datasources.statistics-
enabled:${wildfly.statistics-enabled:false}}" />
   </feature>
```

3. カスタム機能パックの構築に使用される **pom.xml** ファイルでは、JBoss EAP の依存関係を参照してください。

```
<dependency>
  <groupId>org.jboss.eap</groupId>
  <artifactId>wildfly-ee-galleon-pack</artifactId> (1)
  <version>7.3.0.GA-redhat-00004</version>
  <type>zip</type>
</dependency>
```

(1) JBoss EAP 拡張パック (JBoss EAP XP) を使用する場合は、この要素の値を **wildfly-galleon-pack** にする必要があります。

これらの依存関係は、Red Hat Maven リポジトリー https://maven.repository.redhat.com/ga/ から入手できます。

4. Galleon Maven プラグインの **build-user-feature-pack** ゴールを使用してカスタムレイヤーを構築します。

関連情報

ベースレイヤー

WildFly Galleon Maven Plugin Documentation

ドライバーおよびデータソースのパッケージ化を Galleon レイヤーとして示す例

4.3.2. JBoss EAP のカスタムプロビジョニングファイル

カスタムプロビジョニングファイルは、**galleon** サブディレクトリーに保存されている **provisioning.xml** というファイル名の XML ファイルです。

以下のコードは、カスタムプロビジョニングファイルを示しています。

```
<?xml version="1.0" ?>
<installation xmlns="urn:jboss:galleon:provisioning:3.0">
  <feature-pack location="eap-s2i@maven(org.jboss.universe:s2i-universe)"> (1)
    <default-configs inherit="false"/> (2)
    <packages inherit="false"/> (3)
  </feature-pack>
  <feature-pack location="com.example.demo:my-galleon-feature-pack:1.0
"> (4)
    <default-configs inherit="false"/>
    <packages inherit="false"/>
  </feature-pack>
  <config model="standalone" name="standalone.xml"> (5)
    <layers>
      <include name="cloud-server"/>
      <include name="my-custom-driver"/>
      <include name="my-custom-datasource"/>
    </layers>
  </config>
  <options> (6)
```

```
    <option name="optional-packages" value="passive+"/>
  </options>
</installation>
```

1. この要素は、現在の eap-s2i feature-pack 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。ビルダーイメージには単一の機能パックのみが含まれていることに注意してください。
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。
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、**com.example.demo:my-Galleon-feature-pack:1.0** 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、**cloud-server** ベースレイヤーと、 **com.example.demo:my-galleon-feature-pack:1.0** 機能パックの **my-custom-driver** および **my-custom-datasource** カスタム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。

4.3.3. ユーザー開発のレイヤーでプロビジョニングされるアプリケーションのビルド

カスタムプロビジョニングファイルを含むディレクトリーからアプリケーションを構築する場合、S2I ビルドプロセスはプロビジョニングファイルを検出し、指示どおりに JBoss EAP サーバーをプロビジョニングします。

前提条件

- ユーザーが開発したレイヤーはアクセス可能な Maven リポジトリーに存在する必要があります。
- アプリケーションディレクトリーには、ユーザーが開発したレイヤーと、それらのレイヤーを含む機能パックを参照する有効なプロビジョニングファイルが含まれている必要があります。

手順

- 標準の S2I ビルドコマンドを入力してアプリケーションをビルドします。
  たとえば、以下のカスタムプロビジョニングファイルをアプリケーションディレクトリーに作成することを想定します。

```
<?xml version="1.0" ?>
<installation xmlns="urn:jboss:galleon:provisioning:3.0">
  <feature-pack location="eap-s2i@maven(org.jboss.universe:s2i-universe)">
    <default-configs inherit="false"/>
    <packages inherit="false"/>
  </feature-pack>
  <feature-pack location="com.example.demo:my-galleon-feature-pack:1.0">
    <default-configs inherit="false"/>
    <packages inherit="false"/>
  </feature-pack>
```

```
    <config model="standalone" name="standalone.xml">
      <layers>
        <include name="cloud-server"/>
        <include name="my-custom-driver"/>
        <include name="my-custom-datasource"/>
      </layers>
    </config>
    <options>
      <option name="optional-packages" value="passive+"/>
    </options>
</installation>
```

以下のコマンドは、**com.example.demo:my-Galleon-feature-pack:1.0** 機能パックを使用してアプリケーションを構築します。これには、**my-custom-driver** および **my-custom-datasource** レイヤーが含まれます。生成されるアプリケーションの名前は **eap-my-custom-db** です。データベースへの接続は環境変数を使用して設定されます。

```
oc build my-app \
-e DEMO_DB=demo \
-e DEMO_PASSWORD=demo \
-e DEMO_HOST=127.0.0.1  \
-e DEMO_PORT=5432 \
-e DEMO_USER=demo \
eap-my-custom-db
```

ユーザー **demo** とパスワード **demo** を使用して、ポート 5432 でデータベースにログインできます。

関連情報

JBoss EAP のカスタムプロビジョニングファイル

第5章 OPENSHIFT 4 の JBOSS EAP イメージストリームからの、EAP73 IMAGESTREAM へのアプリケーションの移行

eap71 および **eap72** イメージストリーム用に開発されたアプリケーションでは、**eap73** イメージストリームで正しく機能するために変更が必要になります。

5.1. EAP73 IMAGESTREAM 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 3.11 上で実行している **eap72** イメージから **eap73** イメージに移行する際に調整される必要があります。

eap72 イメージでは、liveness プローブのデフォルトの **YAML** 設定は以下のコード例と同等のものです。

OpenShift 3.11 liveness プローブでの eap72 イメージの 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。

プローブに 3 回失敗すると、コンテナーは正常でないとみなされ、OpenShift は Pod でコンテナーを再起動します。

eap72 イメージでは、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。この呼び出しの後に 10 秒の待機時間が続きます。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。

eap73 イメージでは、単一の呼び出し時間は 1 秒未満です。3 つの呼び出しは約 23 秒続きます。**eap73** 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。

eap73 イメージストリーム liveness プローブの YAML 設定例

```
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 **eap72** の動作とほぼ同等で約 34 秒続きます。

readiness プローブでは、**YAML** 設定の **periodSeconds** を同様の値で更新します。

eap73 イメージストリーム readiness プローブの YAML 設定例

```
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

Monitoring Container Health on OpenShift 4

Liveness and Readiness Probes on OpenShift 3.11

5.2. デフォルトのデータソースが削除されました

JBoss EAP 7.3 では、デフォルトのデータソースが JBoss EAP イメージストリームから削除されています。

デフォルトのデータソースを使用するカスタムアプリケーションを開発した場合は、サーバーのプロビジョニング時に、このカスタムアプリケーションを含めることができます。**ENABLE_GENERATE_DEFAULT_DATASOURCE** 環境変数を **true** の値として使用します。

```
ENABLE_GENERATE_DEFAULT_DATASOURCE=true
```

5.3. OPENSHIFT 上の JBOSS EAP 7.1 を JBOSS EAP 7.3 にアップグレードするときに STANDALONE-OPENSHIFT.XML へ更新

JBoss EAP 7.1 でインストールされた設定ファイル **standalone-openshift.xml** は、JBoss EAP 7.3 以降と互換性がありません。

JBoss EAP 7.1 から JBoss EAP 7.3 にアップグレードした後に **standalone-openshift.xml** ファイルを使用する場合は、ファイルに以下の変更を加える必要があります。

- **logging** サブシステムのバージョンを更新します。
  を

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

  上記の行を、以下のように置き換えます。

  ```
  <subsystem xmlns="urn:jboss:domain:logging:8.0">
  ```

- **logging** サブシステム設定のログフォーマッターを更新します。
  を

```
<custom-formatter module="org.jboss.logmanager.ext"
class="org.jboss.logmanager.ext.formatters.LogstashFormatter">
    <properties>
        <property name="metaData" value="log-handler=CONSOLE"/>
    </properties>
</custom-formatter>
```

上記の行を、以下のように置き換えます。

```
<json-formatter>
    <exception-output-type value="formatted"/>
    <key-overrides timestamp="@timestamp"/>
    <meta-data>
        <property name="@version" value="1"/>
    </meta-data>
</json-formatter>
```

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
   ```

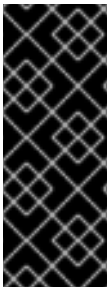
   b. OpenShift イベントを表示するには、以下のコマンドを使用します。

   ```
   $ 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 テクノロジープレビュー機能のサポート範囲 を参照してください。

注記

このチュートリアルでは、OpenShift 3 と、EAP オペレーターを使用せずにトランザクションを復元する場合にのみ該当します。

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。このチュートリアルでは、**jta-crash-rec-eap** クイックスタートサンプルと **eap73-tx-recovery-s2i** アプリケーションテンプレートを使用して、クラスターのスケールダウンによって OpenShift Pod で終了されたときに XA トランザクションがどのように専用の移行 Pod によってリカバリーされるかを示します。JDK 11 イメージのアプリケーションテンプレートは **eap73-openjdk11-tx-recovery-s2i** です。

このチュートリアルでは、**amq-broker-72-openshift:1.1** イメージストリームを使用して JMS 準拠のメッセージブローカーを提供します。初期ブローカー Pod を設定すると、OpenShift Container Platform 機能を使用してクイックスタートを迅速にデプロイできます。

注記

クイックスタートを使用するための前提条件として、AMQ Long Term Support (LTS)イメージからイメージストリームを作成し、イメージストリーム **amq-broker-72-openshift:1.1** という名前を指定する必要があります。Red Hat Ecosystem Catalog から **amq7/amq-broker-lts-rhel7:7.4** コンテナーイメージをダウンロードして LTS イメージにアクセスできます。AMQ Broker のインストールに関する詳細は、基本ブローカーのデプロイ を参照してください。

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。

その他のリソース

- 基本的な AMQ Broker の作成に関する詳細は、**OpenShift Container Platform での AMQ Broker のデプロイ** の 基本的なブローカーのデプロイ を参照してください。

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 eap73-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
```

5. 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 **eap73-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. **eap73-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: eap73-tx-recovery-s2i アプリケーションテンプレート (JDK 8)

   ```
   $ oc new-app --template=eap73-tx-recovery-s2i \
   --name=eap-app
   ```

   例: eap73-openjdk11-tx-recovery-s2i アプリケーションテンプレート (JDK 11)

   ```
   $ oc new-app --template=eap73-openjdk11-tx-recovery-s2i \
   --name=eap-app
   ```

2. 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。

3. **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-amq       1          1         1         config,image(amq-broker-72-openshift:1.1)
   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-amq-1-mws7x     1/1      Running    0          2m
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-amq-1-mws7x         1/1       Running       0          3m
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 OPENSHIFT でのアプリケーションディプロイメントを自動化する EAP OPERATOR

EAP オペレーターは、OpenShift API を拡張する JBoss EAP 固有のコントローラーです。EAP オペレーターを使用することで、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 オペレーターは、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 オペレーターは、EJB 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 オペレーターは OperatorHub を使用して EAP オペレーターをインストールする必要があります。これは、OpenShift クラスター管理者がオペレーターの検出、インストール、アップグレードに使用できます。

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。

8.1. 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 オペレーターをインストールできます。その後、EAP オペレーターを複数の名前空間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 オペレーター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。

- チャンネルの更新: EAP オペレーター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、これを一覧から選択します。

- 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** 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。

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 オペレーターを見つけます。
3. JBoss EAP オペレーターを選択し、**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。

7. **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。

8.2. 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 オペレーターをインストールできます。その後、EAP オペレーターを複数の名前空間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 オペレーター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 の一覧を表示します。

```
$ 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、名前空間を EAP オペレーター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: alpha
  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 オペレーターは **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 オペレーターが正常にインストールされます。この時点で、OLM は EAP のオペレーターを認識します。オペレーターの ClusterServiceVersion (CSV) がターゲット名前空間に表示され、EAP オペレーターによって提供される API は作成に利用できます。

8.3. EAP OPERATOR を使用した OPENSHIFT での JAVA アプリケーションデプロイ

EAP オペレーターを使用することで、OpenShift での Java アプリケーションのデプロイメントを自動化できます。EAP オペレーター API の詳細は、EAP Operator: API Information を参照してください。

OpenShift での Java アプリケーションのデプロイには、以下のアプリケーションイメージタイプのいずれかを選択できます。

- ビルダーイメージまたはランタイムイメージをベースとするアプリケーションイメージ。**eap-s2i-build** テンプレートを使用して、このようなイメージを準備できます。
- ベースイメージレジストリー **registry.access.redhat.com/ubi8/openjdk-11** や をベースとする起動可能な JAR アプリケーションイメージ、またはより高い JDK バージョンを提供するその他の Red Hat ubi8。

EAP オペレーターを使用して OpenShift に Java アプリケーションをデプロイする場合は、いくつかの設定が必要になります。他の一部の設定は、アプリケーションの EAP オペレーター CustomResource(CR) が **Secret** オブジェクトまたは **ConfigMap** を参照する場合にのみ必要です。

その他のリソース

- **eap-s2i-build** テンプレートの詳細は、アプリケーションイメージを作成するための eap-s2i-build テンプレート を参照してください。
- **eap-s2i-build** テンプレートを使用してアプリケーションイメージを構築する方法は、**eap-s2i-build** テンプレートを使用したアプリケーションイメージのビルド を参照してください。
- 起動可能な JAR アプリケーションイメージの使用に関する詳細は、Bootable JAR for packaging EAP server and Java application を参照してください。
- アプリケーションイメージを起動可能な JAR としてパッケージ化する方法は、JBoss EAP OpenShift プラットフォームでの起動可能な JAR の使用 を参照してください。
- Java アプリケーションの OpenShift へのデプロイ時に必須の設定を完了する方法は、Deploying a Java application using the EAP operator: Completing mandatory configurations を参照してください。

- Java アプリケーションの OpenShift へのデプロイ時に任意の設定を完了する方法は、Deploying a Java application using the EAP operator:Completing the optional configurations を参照してください。

8.3.1. アプリケーションイメージを作成するための eap-s2i-build テンプレート

eap-s2i-build テンプレートを使用してアプリケーションイメージを作成します。**eap-s2i-build** テンプレートは複数のパラメーターを追加して、アプリケーションのビルドに使用するアプリケーションソースリポジトリーの場所と EAP S2I イメージを設定します。

eap-s2i-build テンプレートの **APPLICATION_IMAGE** パラメーターは、アプリケーションイメージに対応するイメージストリームの名前を指定します。たとえば、**eap-s2i-build** テンプレートから **my-app** という名前のアプリケーションイメージを作成した場合には、**my-app** イメージ ストリームから **my-app:latest** イメージストリームタグを使用してアプリケーションをデプロイすることができます。**eap-s2i-build** テンプレートで使用されるパラメーターの詳細は、Building an application image using eap-s2i-build template を参照してください。

このテンプレートを使用すると、EAP オペレーターは OpenShift にデプロイされたアプリケーションをシームレスにアップグレードできます。シームレスなアップグレードを有効にするには、GitHub リポジトリーで Webhook を設定し、ビルド設定で Webhook を指定する必要があります。webhook は、リポジトリーが更新され、新規ビルドがトリガーされる際に OpenShift に通知します。

このテンプレートを使用して、JBoss EAP 7.3、JBoss EAP XP、JBoss EAP CD などの JBoss EAP バージョンのイメージストリームを使用してアプリケーションイメージをビルドできます。

その他のリソース

- **eap-s2i-build** テンプレート を使用してアプリケーションイメージをビルドします。

8.3.2. eap-s2i-build テンプレートを使用したアプリケーションイメージのビルド

eap-s2i-build テンプレートは複数のパラメーターを追加して、アプリケーションのビルドに使用するアプリケーションソースリポジトリーの場所と EAP S2I イメージを設定します。このテンプレートを使用すると、JBoss EAP 7.3、JBoss EAP XP、または JBoss EAP CD などのすべての JBoss EAP バージョンにイメージストリームを使用できます。

手順

1. EAP イメージを OpenShift にインポートします。詳細は、JBoss EAP XP の最新の 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 を参照してください。

2. アプリケーションイメージストリームの変更についての更新を受信し、新規ビルドをトリガーするようにイメージストリームを設定します。詳細は、イメージストリームタグの定期的なインポートの設定 を参照してください。

3. EAP S2I イメージを使用してアプリケーションイメージをビルドするための **eap-s2i-build** テンプレートを作成します。

```
$ oc replace --force -f https://raw.githubusercontent.com/jboss-container-images/jboss-eap-openshift-templates/master/eap-s2i-build.yaml
```

この **eap-s2i-build** テンプレートは 2 つのビルド設定と、中間ビルドアーティファクトと最終的なアプリケーションイメージに対応する 2 つのイメージストリームを作成します。

4. 最終的なアプリケーションイメージのリソースを作成するために、パラメーターとともに **eap-s2i-build** テンプレートを処理します。以下の例は、アプリケーションイメージ **my-app** を作成します。

```
$ oc process eap-s2i-build \
  -p APPLICATION_IMAGE=my-app \ (1)
  \
  -p EAP_IMAGE=jboss-eap-xp1-openjdk11-openshift:1.0 \ (2)
  -p EAP_RUNTIME_IMAGE=jboss-eap-xp1-openjdk11-runtime-openshift:1.0 \ (3)
  -p EAP_IMAGESTREAM_NAMESPACE=$(oc project -q) \ (4)
  \
  -p SOURCE_REPOSITORY_URL=https://github.com/jboss-developer/jboss-eap-
quickstarts.git \ (5)
  -p SOURCE_REPOSITORY_REF=xp-1.0.x \ (6)
  -p CONTEXT_DIR=microprofile-config | oc create -f - (7)
```

1. アプリケーションイメージストリームの名前。アプリケーションイメージは **latest** でタグ付けされます。
2. EAP ビルダーイメージのイメージストリームタグ。
3. EAP ランタイムイメージのイメージストリームタグ。
4. Red Hat ミドルウェアイメージのイメージストリームがインストールされている名前空間。省略されている場合、**openshift** 名前空間が使用されます。これは、**openshift** 以外の名前空間にイメージストリームをインストールしている場合にのみ変更します。
5. アプリケーションの Git ソース URL。
6. Git ブランチ/タグリファンレンス
7. ビルドするアプリケーションが含まれる Git リポジトリー内のパス。

5. EAP オペレーターを使用して、デプロイメント用にアプリケーションイメージを準備します。

   a. **WildFlyServer** リソースを設定します。

```
$ cat > my-app.yaml<<EOF

apiVersion: wildfly.org/v1alpha1
kind: WildFlyServer
metadata:
  name: my-app
spec:
 applicationImage: 'my-app:latest'
 replicas: 1
EOF
```

   b. 設定を適用し、EAP オペレーターがこのアプリケーションイメージを参照する新しい **WildFlyServer** リソースを作成できるようにします。

```
$ oc apply -f my-app.yaml
```

   c. 以下のコマンドを使用して、**WildFlyServer** リソースを表示します。

```
$ oc get wfly my-app
```

関連情報

- アプリケーションイメージストリームのインポートに関する詳細は、JBoss EAP XP の最新の 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 を参照してください。
- イメージストリームの定期的なインポートについての詳細は、イメージストリームタグの定期的なインポートの設定 を参照してください。

8.3.3. JBoss EAP サーバーと Java アプリケーションをパッケージ化するための起動可能な JAR

JBoss EAP サーバーおよび Java アプリケーションを実行可能な JAR ファイルとしてパッケージ化できます (起動可能な JAR とも呼ばれます)。この起動可能な JAR を使用すると、サーバー、パッケージ化されたアプリケーション、サーバーの起動に必要なランタイムが含まれる起動可能な JAR アプリケーションイメージをビルドできます。このようにビルドされた起動可能な JAR アプリケーションイメージは、EAP オペレーターを使用して OpenShift にデプロイできます。

EAP オペレーターを使用して、起動可能な JAR イメージを OpenShift にデプロイするには、ベースイメージ **registry.access.redhat.com/ubi8/openjdk-11** を使用するか、より高い JDK バージョンを提供するその他の Red Hat ubi8 を使用する必要があります。

起動可能な JAR は、クラウド環境に設定されたサーバー用に構築する必要があります。**wildfly-jar-maven-plugin** 設定で有効にすると、クラウド環境のサーバーを設定できます。

JBoss EAP OpenShift プラットフォームでの起動可能な JAR の使用 セクションの手順 1 から 6 を実施して、起動可能な JAR アプリケーションイメージをビルドします。これらの手順を完了すると、起動可能な JAR アプリケーションイメージは OpenShift でイメージストリームとして利用でき、EAP オペレーター設定で imagestreamtag を使用できます。

注記

EAP オペレーターは、起動可能な JAR アプリケーションイメージを実行する Pod がスケールダウンすると、トランザクションをリカバリーしません。EAP オペレーターは、Pod のスケールダウン時にトランザクションを復元する可能性を記述するトレースをログに記録します。

その他のリソース

- 起動可能な JAR の詳細は、起動可能な JAR について を参照してください。
- クラウド環境のサーバーの設定に関する詳細は、OpenShift の起動可能な JAR の設定 を参照してください。
- OpenShift への起動可能な JAR アプリケーションイメージのデプロイに関する詳細は、EAP オペレーターを使用した Java アプリケーションの展開: 必須の設定 を参照してください。

8.3.4. EAP オペレーターを使用した Java アプリケーションのデプロイ: 必須の設定

EAP オペレーターを使用して OpenShift に Java アプリケーションをデプロイする場合、以下の設定は必須です。

要件

要件

- EAP オペレーターがインストールされている。EAP オペレーターのインストールに関する詳細は、Installing EAP Operator Using the webconsole 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。

eap-s2i-build テンプレートを使用してアプリケーションイメージをビルドする場合:

- JBoss EAP for OpenShift Source-to-Image (S2I) ビルダー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。
- OpenShift へのデプロイ後にアプリケーションの自動アップグレードを有効にする場合には、**eap-s2i-build** テンプレートの **APPLICATION_IMAGE** パラメーターにイメージストリームがあります。**eap-s2i-build** テンプレートを使用してアプリケーションイメージを構築する方法は、eap-s2i-build テンプレートを使用したアプリケーションイメージのビルド を参照してください。

起動可能な JAR アプリケーションイメージを使用する場合:

- ベースイメージ **registry.access.redhat.com/ubi8/openjdk-11** や、より高い JDK バージョンを提供するその他の Red Hat ubi8 を使用して、起動可能な JAR アプリケーションイメージをビルドしている。
- クラウド環境用にサーバーを設定している。

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 オペレーター**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。**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以下のような例で指定できます。

   - 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
   - イメージハッシュ: quay.io/bootable-jar/myapp@sha256:47c06c96e80d0defb777686cdb468c636d9b3b7081a35b784330a050a
6. アプリケーションのサイズを指定します。例を以下に示します。

```
spec:
  replicas:2
```

- **オプション**: アプリケーションのパッケージ化に起動可能な JAR を使用している場合は、以下の例のように表示されます。

```
spec:
  bootableJar: true
```

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 などの値や POSTGRESQL_USER などの **Secret** オブジェクトから直接取得できます。例を以下に示します。

```
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
```

その他のリソース

- Java アプリケーションの OpenShift へのデプロイ時に任意の設定を完了する方法は、EAP オペレーターを使用した Java アプリケーションの展開: オプションの設定 を参照してください。
- **eap-s2i-build** テンプレートでアプリケーションイメージを構築する方法は、**eap-s2i-build** テンプレートを使用したアプリケーションイメージのビルド を参照してください。
- アプリケーションイメージを起動可能な JAR としてパッケージ化する方法は、JBoss EAP OpenShift プラットフォームでの起動可能な JAR の使用 を参照してください。
- 環境変数の一覧は、環境変数 を参照してください。

8.3.5. EAP オペレーターを使用した Java アプリケーションのデプロイ: 任意の設定

アプリケーションの EAP オペレーター CustomResource (CR) が **Secret** オブジェクトまたは **ConfigMap** を参照する場合は、EAP オペレーターを使用してアプリケーションを OpenShift にデプロイする際に以下のオプションの設定を実行します。

注記

JBoss EAP 7 では、**ConfigMap** から **standalone.xml** ファイルを指定できません。

要件

- EAP オペレーターがインストールされている。EAP オペレーターのインストールに関する詳細は、Installing EAP Operator Using the webconsole 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。
- OpenShift での Java アプリケーションのデプロイに必須の設定を完了している。
- アプリケーションの EAP オペレーター CR が参照する場合は、**Secret** オブジェクトを作成している。**Secret** オブジェクト作成の詳細は Creating a Secret を参照してください。
- アプリケーションの EAP オペレーター CR が参照する場合、**ConfigMap** を作成している。**ConfigMap** 作成の詳細は Creating a ConfigMap を参照してください。
- **オプション**: **standalone.xml** ファイルから **ConfigMap** を作成している。**standalone.xml** ファイルからの **ConfigMap** の作成の詳細は、Creating a ConfigMap from a standalone.xml File を参照してください。

手順

- 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。

  a. 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。

  b. **WildFlyServerSpec** で作成した **Secret** の名前を指定することで、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 **Secret** オブジェクトをマウントします。例を以下に示します。

  ```
  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

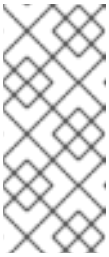
Secret オブジェクトを変更すると、プロジェクトの一貫性が失われることがあります。プロジェクトの不整合を回避するには、古いオブジェクトと同じコンテンツを持つ新しい **Secret** オブジェクトを作成します。これで、必要に応じてコンテンツを更新し、EAP オペレーターカスタムリソース (CR) の参照を更新できます。これは新しい CR 更新とみなされ、pod はリロードされます。

c. **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
```

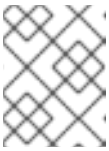


注記

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。プロジェクトの一貫性を避けるために、古い内容と同じコンテンツを持つ新規 **ConfigMap** を作成します。これで、必要に応じてコンテンツを更新し、EAP 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。

d. 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-openshift.xml
```



注記

JBoss EAP 7 では、**standalone.xml** ファイルから **ConfigMap** を作成することはできません。

e. 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
```

その他のリソース

- サーバーデータディレクトリーのストレージ要件の指定に関する詳細は、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。
- Java アプリケーションの OpenShift へのデプロイ時に必須の設定を完了する方法は、EAP オペレーターを使用した OpenShift での Java アプリケーションのデプロイ を参照してください。

8.3.6. Secret の作成

アプリケーションの EAP オペレーター CustomResource (CR) が **Secret** を参照する場合は、EAP オペレーター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** オブジェクト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'
```

8.3.7. ConfigMap の作成

アプリケーションの EAP operator CustomResourceDefinition (CRD) が **spec.ConfigMaps** フィールドの **ConfigMap** を参照する場合は、EAP オペレーター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
```

8.3.8. 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

注記: JBoss EAP 7 では、**ConfigMap** から **standalone.xml** ファイルを指定することはできません。

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-openshift.xml
configmap/clusterbench-config-map created
```

8.3.9. 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。

8.4. EAP オペレーターを使用したアプリケーションのメトリクスの表示

EAP オペレーターを使用すると、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 オペレーターは OpenShift コンソールでメトリクスを自動的に表示します。

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. **メトリクス** 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。

注記

JBoss EAP アプリケーションサーバーに関連するすべてのメトリクスの前に **jboss** が付けられます。例: **jboss_undertow_request_count_total**

8.5. WEB コンソールを使用した EAP OPERATOR のアンインストール

クラスターから EAP オペレーターを削除またはアンインストールするには、サブスクリプションを削除して、サブスクライブされた名前空間から削除してください。EAP オペレーターの 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には、クラスターでの pod 数を 0 に下げてから EAP オペレーターをアンインストールします。

Web コンソールを使用して EAP オペレーターを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 EJB 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 オペレーターは実行を停止し、更新を受信しなくなります。

8.6. CLI を使用した EAP OPERATOR のアンインストール

クラスターから EAP オペレーターを削除またはアンインストールするには、サブスクリプションを削除して、サブスクライブされた名前空間から削除してください。EAP オペレーターの 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には、クラスターでの pod 数を 0 に下げてから EAP オペレーターをアンインストールします。

EAP オペレーターは、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 EJB 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** フィールドの EAP オペレーター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 オペレーター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. 前回の手順の **currentCSV** 値を使用し、ターゲット名前空間で EAP オペレーターの 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
```

8.7. 安全なトランザクションリカバリーの EAP OPERATOR

EAP オペレーターは、アプリケーションクラスターを削除する前にデータの一貫性を確立します。これは、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行います。

これは、データが整合した状態でディプロイメントを安全に削除するのであれば、最初に Pod の数を 0 に縮小し、すべての pod が削除されるまで待ってからはじめて **wildflyserver** インスタンスを削除するということを意味します。

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 EJB 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 EJB 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、Monitoring the Scaledown Process を参照してください。縮小を行う際の Pod ステータスの詳細は、Pod Status During scaleDown を参照してください。

注記

EAP オペレーターは、起動可能な JAR アプリケーションイメージを実行する Pod がスケールダウンすると、トランザクションをリカバリーしません。EAP オペレーターは、Pod のスケールダウン時にトランザクションを復元する可能性を記述するトレースをログに記録します。

8.7.1. 安定したネットワークホスト名の StatefulSets

wildflyserver を管理する EAP オペレーターは、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。

8.7.2. Scaledown プロセスの監視

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する必要があります。縮小時の各種 Pod ステータスの詳細は、Pod Status During scaleDown 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。

8.7.2.1. 縮小中の 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8.1 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。

8.7.3. 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 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. EJB 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 EJB 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。

8.7.4. 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

- データソース設定およびトランザクションサブシステム設定の両方を確認するには、**standalone-openshift.xml** 設定ファイル **oc rsh <podname> cat /opt/eap/standalone/configuration/standalone-openshift.xml** を確認します。
  想定される出力:

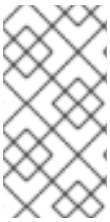
```
<datasource jta="false" jndi-name="java:jboss/datasources/PostgresJdbcObjectStore" pool-name="postgresjdbcobjectstore_postgresqlObjectStorePool"
    enabled="true" use-java-context="true" statistics-enabled="${wildfly.datasources.statistics-enabled:${wildfly.statistics-enabled:false}}">
    <connection-url>jdbc:postgresql://postgresql:5432/sampledb</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**設定ガイド**の データソースの作成 を参照してください。

8.8. OPENSHIFT 上の EJB リモーティング

JBoss EAP で、OpenShift 上の各種 JBoss EAP クラスター間で EJB リモーティング呼び出しを正しく機能させるには、OpenShift の EJB 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 へのデプロイ時に、EAP オペレーターの使用を考慮してください。EAP オペレーターは、EJB 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 EJB 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。

EJB 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、リモート EJB の使用 を参照してください。

EJB 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 EJB 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 オペレーターは、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 オペレーターを使用する場合、ヘッドレスサービスは **eap-server-headless** といった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、EJB 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、EJB 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。

8.8.1. OpenShift での EJB の設定

EJB 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、EJB 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。

要件

- EAP オペレーターと、対応の JBoss EAP for OpenShift S2I イメージ使用して、OpenShift で JBoss アプリケーションインスタンスのディプロイメントと管理を行っている。
- クラスターリングが正しく設定されている。JBoss EAP クラスターリングの詳細は、クラスターリング セクションを参照してください。

手順

手順

1. EJB 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9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。

9.1. 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。

9.2. 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。

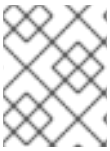
表9.1 情報環境変数

変数名	説明および値
JBOSS_IMAGE_NAME	イメージ名 値: ● **jboss-eap-7/eap73-openjdk8-openshift-rhel7** (JDK 8 / RHEL 7) ● **jboss-eap-7/eap73-openjdk11-openshift-rhel8** (JDK 11 / RHEL 8)

変数名	説明および値
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 8 / RHEL 7 ● JDK 11 / 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: **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**

9.3. 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。



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9.2 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。

変数名	説明
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**
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, ...** の形式で表示される必要があります。 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**

変数名	説明
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**
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**
AB_PROMETHEUS_ENABLE	**true** に設定すると、この値により、Prometheus 形式のメトリクスを公開する **jmx-exporter** java エージェントが有効になります。デフォルトは **false** に設定されます。 **注記** MicroProfile Metrics サブシステムは、データを Prometheus 形式で公開するための推奨される方法です。MicroProfile Metrics サブシステムの詳細は、JBoss EAP**設定ガイド** の Eclipse MicroProfile を参照してください。
AB_PROMETHEUS_JMX_EXPORTER_CONFIG	**jmx-exporter** エージェントがによりデフォルトの **configuration.yaml** ファイルの代わりに使用する、ユーザーが定義した **configuration.yaml** に対するコンテナー内のパス。追加の設定ファイルを取り入れる **S2I** メカニズムの詳細は、S2I アーティファクト を参照してください。
AB_PROMETHEUS_JMX_EXPORTER_PORT	**jmx-exporter** エージェントが Prometheus サーバーからスクレープをリッスンするポート。デフォルト値は **9799** です。エージェントは **localhost** をリッスンします。アプリケーションに、このエンドポイントを公開するサービスを含めるように **DeploymentConfig** ファイルを設定することで、メトリックをコンテナーの外部で使用できるようにできます。
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**

変数名	説明
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**
DISABLE_EMBEDDED_JMS_BROKER	OpenShift コンテナーでの組み込みメッセージングブローカーの使用は非推奨となりました。組み込みブローカーのサポートは今後のリリースで削除されます。 以下の条件が true の場合は、警告がログに記録されます。 ● コンテナーは、埋め込みメッセージングブローカーを使用するよう設定されます。 ● リモートブローカーはコンテナー用に設定されていません。 ● この変数が設定されていないか、**false** の値で設定されます。 この変数が **true** に設定された値に含まれる場合は、埋め込みメッセージングブローカーは無効になり、警告はログに記録されません。 リモートメッセージング宛先で設定されていないコンテナーの場合は、この変数を **true** に追加します。
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。

変数名	説明
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**
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**

変数名	説明
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_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。**dns.DNS_PING** または **kubernetes.KUBE_PING** のいずれかを使用できます。
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ーリングラベルセレクター。 値の例: **app=eap-app**

変数名	説明
OPENSHIFT_KUBE_PING_NAMESPACE	Kubernetes 検索メカニズムのクラスター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。

9.4. アプリケーションテンプレート

表9.3 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**

9.5. 公開されたポート

表9.4 公開されたポート

ポート番号	説明
8443	HTTPS
8778	Jolokia の監視

9.6. 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。

9.6.1. 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。

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お勧めします。

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。

9.6.1.1. 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、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**
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**

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。

9.6.1.2. 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。

9.6.1.2.1. 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。

9.6.1.2.2. 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。

9.7. クラスタリング

9.7.1. JGroups 検索メカニズムの設定

OpenShift で JBoss EAP クラスター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-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 **dns.DNS_PING** になります。

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。

9.7.1.1. 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。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。

9.7.1.2. 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 **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。

9.7.2. 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。

9.7.2.1. 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。以下に例を示します。

   ```
   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。以下に例を示します。

   ```
   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。以下に例を示します。

```
JGROUPS_ENCRYPT_KEYSTORE=jgroups.jceks
```

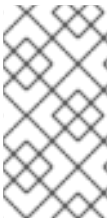
5. **JGROUPS_ENCRYPT_NAME** 環境変数を、サーバーの証明書に関連する名前に設定する必要があります。設定しないと、クラスター通信は暗号化されず、警告が発生します。以下に例を示します。

```
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。以下に例を示します。

```
JGROUPS_ENCRYPT_PASSWORD=mypassword
```

9.7.2.2. ASYM_ENCRYPT の設定



注記

JBoss EAP 7.3 には **ASYM_ENCRYPT** プロトコルの新しいバージョンが含まれています。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 クラスター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_SECRET="encrypt_secret" \
-e JGROUPS_ENCRYPT_NAME="encrypt_name" \
-e JGROUPS_ENCRYPT_PASSWORD="encrypt_password" \
-e JGROUPS_ENCRYPT_KEYSTORE="encrypt_keystore" \
-e JGROUPS_CLUSTER_PASSWORD="cluster_password"
```

9.8. 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9.5 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。

9.9. MESSAGING

9.9.1. 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: JDK 8

```
oc new-app eap73-amq-s2i \
-p APPLICATION_NAME=eap73-mq \
-p MQ_USERNAME=MY_USERNAME \
-p MQ_PASSWORD=MY_PASSWORD
```

例: JDK 11

```
oc new-app eap73-openjdk11-amq-s2i \
-p APPLICATION_NAME=eap73-mq \
-p MQ_USERNAME=MY_USERNAME \
-p MQ_PASSWORD=MY_PASSWORD
```

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。

9.10. 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9.6 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。

9.11. 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**

9.12. 管理環境変数

表9.7 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**

9.13. 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。

9.13.1. 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。

9.13.1.1. 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。

9.13.2. 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** はコンマ区切りのリストです。例:

ARTIFACT_DIR=app1/target,app2/target,app3/target

9.13.3. アーティファクトリーポジトリーミラー

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。

9.13.3.1. セキュアなアーティファクトリーポジトリーミラー URL

Maven リポジトリーを使用した "man-in-the-middle" 攻撃を回避するために、JBoss EAP ではアーティファクトリー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。

9.13.4. 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。

9.13.5. 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。

9.13.5.1. 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/eap/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-openshift.xml
:whoami
quit
```

9.13.5.2. 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
   ```

9.13.6. 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9.8 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**
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_DEFAULT_FAT_SERVER	(オプション)**true** の値とともに含まれている場合、galleon レイヤーが設定されていなければ、デフォルトの JBoss EAP サーバーはプロビジョニングされます。
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一覧です。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, sso**
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**

変数名	説明
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ーポジトリーミラー 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**
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。

9.14. シングルサインオンイメージ

このイメージには、Red Hat シングルサインオン対応アプリケーションが含まれます。

JBoss EAP for OpenShift イメージを使用して Red Hat Single Sign-On for OpenShift イメージをデプロイする方法の詳細は、**Red Hat Single Sign-On for OpenShiftガイド**の Deploy the Red Hat Single Sign-On-enabled JBoss EAP Image を参照してください。

表9.9 シングルサインオンの環境変数

変数名	説明
SSO_URL	シングルサインオンサーバーの URL。

変数名	説明
SSO_REALM	デプロイされたアプリケーションのシングルサインオンレルム。
SSO_PUBLIC_KEY	Single Sign-On 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。
SSO_USERNAME	シングルサインオン REST API にアクセスするのに必要なシングルサインオンユーザー。 値の例: **mySsoUser**
SSO_PASSWORD	**SSO_USERNAME** 変数によって定義されたシングルサインオンの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	シングルサインオンクライアントアクセスタイプ。(オプション) 値の例: **true**
SSO_CLIENT	シングルサインオンのパスは、再度アプリケーションにリダイレクトします。デフォルトは **module-name** と一致します。
SSO_ENABLE_CORS	**true** の場合、シングルサインオンアプリケーションの CORS を有効にします。(オプション)
SSO_SECRET	機密アクセスのためのシングルサインオン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 シングルサインオンサーバー間の 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**

9.15. 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。

重要

自動化トランザクションリカバリー機能は OpenShift 3 でのみサポートされ、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。

OpenShift 4 では、EAP Operator を使用してトランザクションを安全にリカバリーできます。安全なトランザクションリカバリーの EAP Operator を参照してください。

9.15.1. 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。

- OpenShift 3 では、JBoss Remoting 上で伝搬されるトランザクションはサポートされません。

注記

JBoss Remoting 上で伝搬されるトランザクションは OpenShift 4 と EAP のオペレーターでサポートされます。

- 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。

9.15.2. 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。

9.15.2.1. 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、**aaa123456789012345678m0jwh** と **bbb123456789012345678m0jwh** は両方 **123456789012345678m0jwh** 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。

9.15.2.2. 前提条件

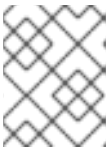
OpenShift インスタンスは JDBC ストアと設定され、ストアテーブルは Pod 名に対応するテーブル接頭辞を使用してパーティションされることを前提とします。これは、JBoss EAP デプロイメントを使用する場合は常に自動である必要があります。これは、共有ボリューム上で split ディレクトリーを持つファイルストアを使用する 自動化リカバリーの例 とは異なります。稼働中の Pod で transaction サブシステムの設定を確認すると、JBoss EAP インスタンスが JDBC オブジェクトストアを使用していることが確認できます。

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。

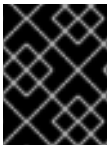
```
<jdbc-store datasource-jndi-name="java:jboss/datasources/jdbcstore_postgresql"/>
```



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。

9.15.2.3. 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。

9.15.2.3.1. 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//rBEAB440GK1aJ72oAAAAGHAtanRhLWNyYXNoLXJlYy0zLXAy
Y2N3_AAAAAAAAAAAAAP//rBEAB440GK1aJ72oAAAAGgAAAAEAAAAA
```

9.15.2.3.2. 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

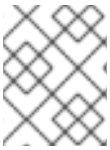
インダウト状態のブランチの XID がすべて特定できたら、XID をトランザクションマネージャーのトランザクションテーブルに保存されたログと比較できる形式に変換します。

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、**os<node-identifier>jbosststxtable** パターンにしたがって命名される JDBC テーブルにあります。
  - 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_AAAAAAAAAAAAAP//rBEAB440GK1aJ72oAAAAGHAtanRh
 ----
LWNyYXNoLXJlYy0zLXAyY2N3_AAAAAAAAAAAAAP//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。

9.15.2.3.3. 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

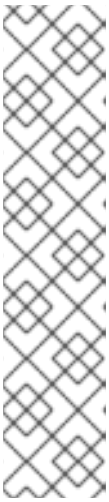
前述の JBoss ノード名識別子 は、常に最大文字数である 23 文字になるよう省略されます。これは、先頭から文字を削除して最大長の 23 文字になるまで末尾の文字を確保します。

9.15.2.3.4. 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)
```

9.15.2.3.5. 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。

9.16. 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9.10 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.openshift.ping
net.oauth.core

9.17. EAP OPERATOR: API 情報

EAP オペレーターにより、以下の API が導入されます。

9.17.1. WildFlyServer

WildFlyServer はカスタム JBoss EAP リソースを定義します。

表9.11 WildFlyServer

フィールド	説明	Scheme	必須
metadata	標準オブジェクトのメタデータ	ObjectMeta v1 meta	false

フィールド	説明	Scheme	必須
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

9.17.2. WildFlyServerList

WildFlyServerList は JBoss EAP デプロイメントのリストを定義します。

表9.12 Table

フィールド	説明	Scheme	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

9.17.3. 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9.13 WildFlyServerSpec

フィールド	説明	Scheme	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

フィールド	説明	Scheme	必須
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一覧	**corev1.EnvFromSource**	false
env	コンテナーに存在する環境変数の一覧	corev1.EnvVar	false
secrets	コンテナーでボリュームとしてマウントされる secret 名の一覧。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** 名の一覧。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

9.17.4. 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9.14 Table

フィールド	説明	Scheme	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

9.17.5. 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9.15 StandaloneConfigMapSpec

フィールド	説明	Scheme	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

フィールド	説明	Scheme	必須
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

9.17.6. 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9.16 WildFlyServerStatus

フィールド	説明	Scheme	必須
replicas	アプリケーションの実際のレプリカ数	int32	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

9.17.7. 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9.17 PodStatus

フィールド	説明	Scheme	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

Revised on 2023-01-28 12:01:17 +1000