MICRO FOCUS

COBOL アプリケーションを コンテナに移行するための ベスト プラクティス

Micro Focus
The Lawn
22-30 Old Bath Road
Newbury, Berkshire RG14 1QN
UK
http://www.microfocus.com

2021 年 4 月 21 日

目次

本書について

本書は、既存の Enterprise Server アプリケーションをコンテナ化された環境に移行してビルドおよび実行する際に Micro Focus が推奨する方法論を示します。

本書で説明している COBOL 関連の機能は、Enterprise Developer 6.0 および Enterprise Server 6.0 で使用できます。COBOL 関連の機能については、すべてではありませんが、一部を Visual COBOL 6.0 および COBOL Server 6.0 で使用できます。

本書の情報は、Enterprise Developer および Visual COBOL のドキュメントを置き換えるものではなく、補足を目的としています。そのため、本書には Enterprise Developer の最新ドキュメントへのリンクが多数含まれています。リンクは Enterprise Developer for Eclipse のドキュメントを対象としているため、Enterprise Developer for Visual Studio または Visual COBOL を使用している場合には、参照先の情報がお使いの製品に該当していることを確認する必要があります。

サード パーティの企業、製品、および Web サイトへの言及は、情報提供のみを目的としたものであり、支持または推奨するものではありません。この情報は、便宜のためにのみ提供されています。Micro Focus は、他の Web サイトのコンテンツまたは当該サード パーティ製品の使用に関していかなる表明もしません。Micro Focus は、当該サード パーティ製品/Web サイトの使用またはアクセスに関して、一切の責任を負わないものとします。

改訂情報

バージョン	日付	変更点
1.0	2020 年 6 月	Visual COBOL および Enterprise Developer 6.0 のリリースに付随するドキュメントの初版
1.1	2020 年 7 月	リソースの安全なスケールダウンに関する追加情報を「*13.1. スケールアウトのディプロイ用にアプリケーションを準備する*」に追記

コンテナ化された環境に移行する利点

コンテナ化された環境でアプリケーションを実行すると、「コンテナを使用する利点」にあるとおり、多くの利点が得られます。コンテナ化された環境で実行する際に最大限の利点を得るには、アプリケーションに加えたすべての変更をバージョン管理システムで追跡できるようにする必要があります。これにより、ディプロイしたアプリケーションのビルドについて十分な追跡可能性および再現性が確保されます。この方法論の要所は、変更点を 1 回につき 1 つとし、アプリケーションで重要な変更を行うごとにアプリケーションを再テストするということです。これにより、発生したエラーをできるだけ早い段階で解決できます。

方法論の概要

方法論は、次に示すように 3 段階に分かれており、各段階はより詳細な手順に分かれています。

段階 A – ソースおよび構成をコードとして保存する

1. 既存のアプリケーション ソース ファイルがすべてバージョン管理システムに保持されていることを確認する
2. 手動での介入を必要とせず、バージョン管理から抽出されたソース ファイルのみを使用する、一貫性のある方法でアプリケーションをビルドできるようにする
3. バージョン管理用にアプリケーション構成を準備する
4. アプリケーション構成をバージョン管理に保存する
5. バージョン管理に保存されているアプリケーション構成を使用して、バージョン管理からビルドしたアプリケーションをテストする

段階 B – アプリケーションをコンテナ化する

6. コンテナを使用してビルドするようにビルド プロセスを変換する
7. コンテナを使用してビルドしたアプリケーションをテストする
8. アプリケーションをコンテナにビルドするようにビルド プロセスを変換する
9. コンテナにビルドしたアプリケーションをテストする

段階 C – スケールアウト環境にディプロイする

10. スケールアウトに対応できるようにデータを変換する
11. スケールアウト環境にディプロイできるようにアプリケーションを構成する
12. スケールアウト環境に保持されているデータを使用して、コンテナにビルドしたアプリケーションをテストする
13. アプリケーション コンテナをスケールアウト環境にディプロイする
14. スケールアウト環境でアプリケーション コンテナのレプリカを複数実行して、アプリケーションをテストする
15. メトリックスを収集および使用して、スケールアウト環境を監視する

各段階について、および各段階を構成する各手順について、以降で説明します。

段階 A – ソースおよび構成をコードとして保存する

本書は全体として、「コンテナを使用する利点」にある利点を得られるように、コンテナ化された環境に移行する方法について説明しています。本セクションでは、コンテナ化された環境で実行するための変更を行う前に、アプリケーションのソース ファイルおよび関連構成ファイルをバージョン管理システムに移動する方法について説明します。

バージョン管理システムの使用は、コンテナ化された環境にアプリケーションを移行する際の要件ではありませんが、Micro Focus では、コンテナ化された環境への移行有無に関わらず、バージョン管理システムの使用を推奨しています。バージョン管理システムを使用すると、アプリケーション ビルドへの変更を追跡して、アプリケーション ライフサイクルの各段階 (開発、QA、本番など) で使用されているバージョンを追跡できるようになります。構成ファイル (およびソース ファイル) をバージョン管理システムに保存すると、アプリケーション プログラムへの変更とまったく同じ方法で、これらのファイルへの変更を管理および追跡できるようになります。

> **注：**ソース ファイルおよび構成ファイルはバージョン管理システムに保存することをお勧めしますが、これは必須ではなく、より簡易な形式であるファイル ストレージの使用も可能です。簡潔さのため、本セクションでは、お使いのバージョン管理システムまたはファイル ストレージの方法を指すものとして「バージョン管理システム」という用語を使用します。
>
> Enterprise Developer 環境でもコンテナ環境でも、バージョン管理システムへのアプリケーション ソース ファイルの保存方法に関して特別な要件は適用されないため、バージョン管理システムを利用する際には通常使用する任意の標準および規則を使用できます。

1.アプリケーションのソース コードをバージョン管理に保存する

アプリケーションのビルドに必要なすべてのソース ファイルがバージョン管理システムに保存されていることを確認します。これには、プログラム ファイル、コピー ファイル、インクルード ファイル、BMS マップセット、および JCL ファイルが含まれますが、これらに限定されません。

アプリケーションのビルドに使用されるスクリプトまたはプロジェクト ファイルも、バージョン管理システムに保存する必要があります。バージョン管理に追加するファイルに、ユーザー資格情報や証明書などのシークレットが含まれていないことを確認してください。

2.アプリケーションをビルドできるようにする

*2.1.*ビルド環境およびランタイム環境を定義する
アプリケーションで使用するビルド環境およびランタイム環境を適切に定義します。これには、アプリケーションのビルドに使用する Visual COBOL/Enterprise Developer のバージョン、アプリケーションの実行に使用する COBOL Server/Enterprise Server のバージョン、およびデータベース プリコンパイラやドライバーなどのその他の製品が含まれます。

ビルド環境およびランタイム環境のこの定義は、残りのソース コードとともにドキュメントとしてバージョン管理に保持する必要があります。このドキュメントは、後でアプリケーションをコンテナ化する際に使用します。

*2.2.*アプリケーションをビルドする
次のみを使用してアプリケーション全体をビルドします。

- 「*2.1.* ビルド環境およびランタイム環境を定義する」で定義したビルド環境
- バージョン管理システムから抽出したファイル

この手順は、インストールされたすべてのソフトウェアに対する緊密な管理を確保できるよう、クリーンなマシンを使用して行うのが望ましいと言えます。

Micro Focus からソース形式で提供されるモジュール (XA スイッチ モジュールなど) および Enterprise Server ユーザー出口 (JCL プリンター出口など) を含む、アプリケーションに必要なすべてのプログラムを漏れなくビルドしてください。

*2.3.*ビルドを自動化する

処理開始以降のユーザー操作を不要にするため、単一のバッチ ファイルまたはシェル スクリプトを使用してアプリケーションをビルドできるようにします。

ディプロイ パッケージを Interface Mapping Toolkit (IMTK) サービス用に生成する必要があります。これは、`imtkmake` コマンド ライン ユーティリティを使用して作成される COBOL アーカイブ (**.car**) ファイルの形式になります。このファイルは、Visual Studio または Eclipse プロジェクト ビルドの一部として統合できます。

3.バージョン管理用にアプリケーション構成を準備する

3.1.Enterprise Server アプリケーションをバージョン管理に保存できるようにする

Enterprise Server では、さまざまなバイナリ ファイルに構成が保存されます。バイナリ ファイルは、2 つのバージョン間での変更を確認しにくいため、バージョン管理システムへの保存には適していません。したがって、できる限りこれらのバイナリ ファイルのテキスト バージョンを作成し、テキスト ファイルをバージョン管理システムに保存してください。

Enterprise Server ユーティリティを使用することで、構成を XML テキスト ファイルにエクスポートできます。これらの XML ファイルのサイズを縮小するには、構成をエクスポートする前に、Enterprise Server の構成について定期メンテナンスおよびクリーンアップを実行する必要があります。また、コンテナ化された環境で簡単に使用できるように、構成を調整する必要があります。次の各セクションで、これらの手順の概要を説明します。

3.2.CICS リソース定義ファイルを準備する

移行先の環境でアプリケーションの実行に使用される Enterprise Server のバージョンが、現在の環境でアプリケーションの実行に使用されているバージョンと異なる場合は、XML への変換前に、アプリケーションで使用する CICS リソース定義ファイルのアップグレードが必要になります。アプリケーションで使用する Enterprise Server のバージョンが同じである場合は、この手順を実行する必要はありません。

方法：Enterprise Developer には、CICS リソース定義ファイルのアップグレードに使用できる `caspcupg` コマンド ライン ユーティリティが用意されています。

> **注：**異なるバージョンの Enterprise Server が移行先となる場合は、アプリケーションを再コンパイル、リビルド、および再テストする必要もあります。これらの作業については、コンテナ化に固有のものではないため、本書では取り上げません。

*3.3.*スプール データ ファイルを準備する

Enterprise Developer では、スプール管理機能を構成する **SPL*** 形式のスプール データ ファイルを多数使用します。これらのファイルをコンテナ化された環境で使用できるようにする前に、ファイルを整理し、関連性がなくなったエントリの削除を検討してください。これは特に、Micro Focus データベース ファイル ハンドラー (MFDBFH) を介してアクセスするファイルを使用してスケールアウト環境に移行する場合に重要です。

この整理を行うには、MVSSPLHK スプール ハウスキーピング プロセスを使用して、最大保持期間を超えたジョブのスプール データ ファイルとジョブ レコードをすべてアーカイブおよび削除します。詳細については、「*MVSSPLHK スプール ハウスキーピング プロセス*」を参照してください。

*3.4.*カタログ ファイルを準備する
スプール データ ファイルの準備と同様に、カタログのメンテナンスを実行して、期限切れのデータセットを削除してください。これにより、不要になったファイルをバックアップおよび復元せずに済みます。

> **注：**必ず、ユーザー カタログのメンテナンスも実施してください。

詳細については、「カタログ メンテナンス」を参照してください。

3.5.JCL カタログ化データセット エントリを準備する
ファイル システムのフル パスを含むカタログ エントリがある場合は、カタログの場所を基準とする相対パスを指定するように変更してください。これにより、リージョンの移植性が向上します。

Enterprise Developer には、カタログ ファイルのこのような更新を一括で実行できる `mvspcrn` コマンドが用意されています。詳細については、「カタログの大量更新」を参照してください。

*3.6.*システム環境変数を確認する
リージョンまたはアプリケーションで使用されている、システム レベルで設定された環境変数を特定する必要があります。これらの変数は Enterprise Server 構成の外部で設定されているため、可能であれば、「ユーザー インターフェイスから環境変数を設定するには」の説明に従い、これらの変数を移動して Enterprise Server で設定します。残りの変数については、コンテナ化された環境で適切に設定できるように文書化する必要があります。

このように Enterprise Server で環境変数を設定すると、リージョンの構成および実行に必要なすべての情報が、リージョン内に保持されることになります。

これを行うには、Enterprise Server の構成で使用されている環境変数を特定します。次に例を示します。

```
$MY-ENV-NAME
```

または

```
$MY-ENV-NAME/myfolder/myfile.txt
```

*3.7.*固定リスナー ポートをリージョンで使用していることを確認する
アプリケーションをコンテナで実行する場合、そのコンテナの外部からアクセスする必要があるリスナーは、固定ポートでリッスンする必要があります。これにより、それらのリスナーをコンテナで明示的に公開できるため、外部からアクセス可能になります。

Enterprise Server Administration インターフェイスを使用して、リージョンのリスナーが固定ポートを使用していること、つまり、***:*** または ***network-adapter:**** としてリストされていないことを確認できます。使用中の固定ポートおよびその番号を控えておいてください。アプリケーションをコンテナ化する際にこの情報が必要になります。

詳細については、「リスナー」および「リスナーのエンドポイント アドレスで固定ポートを設定するには」を参照してください。

*3.8.*シークレットを特定して資格情報コンテナを使用する
この場合のシークレットとは、ユーザー資格情報や証明書など、機密性の高い構成項目を指し、資格情報コンテナとは、Enterprise Server の「コンテナ機能」のことです。「コンテナ機能」により、Enterprise Server コンポーネントで、資格情報コンテナとして定義されたストレージの形式でこのような機密情報を保持でき、構成可能な資格情報コンテナ プロバイダーを

使用してこの情報にアクセスできます。

Enterprise Server で使用される資格情報は資格情報コンテナに格納されるため、「コンテナ機能」をまだ使用していない場合は、この機能を有効にしてください。Micro Focus Directory Server (MFDS) で資格情報コンテナを使用できるようにするには、`MFDS_USE_VAULT=Y` 環境変数を指定します。

資格情報コンテナによって作成されたファイルは、**$COBDIR/etc/secrets.cfg** ファイル (Linux の場合) または **%PROGRAMDATA%\Micro Focus\Enterprise Developer\mfsecrets\secrets.cfg** ファイル (Windows の場合) の情報を使用して暗号化されます。資格情報コンテナによって作成された暗号化ファイルは、資格情報コンテナのファイル システムの場所 (**secrets.cfg** ファイルの `location` 要素で定義) で確認できます。

Enterprise Server および MFDS の構成をエクスポートする際に資格情報コンテナが有効になっていない場合、パスワードは、構成 XML ファイルでプレーン テキストとして表示されます。セキュリティ上の理由から、パスワードをバージョン管理システムにプレーン テキストとして保存しないでください。

詳細については、「コンテナ機能」および「デフォルトのコンテナの構成」を参照してください。

4.アプリケーション構成をバージョン管理に保存する

Enterprise Server 構成の詳細は、アプリケーションの実行時に効率的に処理できるように、バイナリ形式のファイルに保存されます。ただし、バージョン管理システムではテキストベースのファイルの方が扱いやすく、バイナリ ファイルはバージョン管理システムへの保存には適していません。

本セクションでは、Enterprise Server のデータおよび構成ファイルを、バージョン管理システムでの使用に適した形式で作成またはエクスポートするための手順について説明します。構成をバージョン管理に保存する主な利点は次のとおりです。

- アプリケーションの構成ファイルについて、ソース ファイルの場合と同じ追跡可能性および説明可能性が得られる
- 構成の変更が参照先のアプリケーション プログラムとともに保存されるため、一貫性が保たれる

アプリケーションでは、テキストベースの形式に簡単には変換できない、アプリケーション固有の構成ファイルが使用されている場合があります。このようなファイルも、コンテナ化された環境で使用できるようにバージョン管理システムに追加する必要がありますが、それらのファイルに対してはバージョン管理システムの一部の機能を使用できなくなります。

*4.1.*リージョンを *XML* にエクスポートする

次のような各種のユーティリティを使用して、Enterprise Server リージョンの定義を XML に変換できます。

- mfds：リージョンの定義のみを XML にエクスポートします。
- casrdtex：CICS リソース定義ファイルのみから XML にエクスポートします。
- mfcatxml：カタログ ファイルのみから XML にエクスポートします。
- casesxml：カタログおよびリソース定義ファイルの定義に加えて、リージョン定義を XML にエクスポートします。

詳細については、次を参照してください。

- *mfds*
- *casrdtex*
- カタログのインポートおよびエクスポート：インポート エクスポート ユーティリティ
- *casesxml*

casesxml を使用するか、個々のユーティリティを使用するかは、エクスポートするアーティファクトによって異なります。たとえば、カタログ ファイルをエクスポートすることが好ましくない本番リージョンの場合は、mfds および casrdtex を使用するのが最適ですが、テストに使用する自己完結型リージョンの場合は、casesxml を使用することもできます。

> **注：**
>
> `mfds` コマンドを使用する場合は、`/x 5` パラメーターを使用する必要があります。このパラメーターを指定しないと、生成される出力は XML ファイルになりません。
>
> `casrdtex` コマンドを使用する場合は、`/xm` スイッチを使用する必要があります。このスイッチを指定しないと、生成される出力は XML ファイルではなくリソース定義テーブル (**.rdt**) ファイルになります。

アプリケーションのテスト時にテスト環境を一貫性のある方法で設定できると便利なため、バージョン管理へのテスト データの保存について検討することをお勧めします。機密性の高い本番データをバージョン管理に保存しないでください。

5.アプリケーションをテストする

アプリケーションおよび構成ファイルをバージョン管理に保存し、バージョン管理のファイルのみからアプリケーションをビルドした後、リビルドしたアプリケーションをテストして、エラーが生じていないことを確認してください。Micro Focus Unified Functional Testing や Silk Test などの自動テスト ツールを使用すると、アプリケーションのテストが必要な場合に一貫性のある方法で簡単に実行できるテストを作成できます。

詳細については、「*UFT One*」および「*Silk Central*」を参照してください。

段階 B – アプリケーションをコンテナ化する

「段階 *A* – ソースおよび構成をコードとして保存する」で説明している手順では、できる限り多くのアプリケーションおよびその構成をバージョン管理システムに保存し、スクリプトまたはバッチ ファイルを作成して単一のコマンドでバージョン管理システム内のファイルからアプリケーションをビルドできるようにし、アプリケーションをテストして、バージョン管理システムに移行する前と同じように機能することを確認しました。

本セクションでは、プロセスの次の段階、つまり既存のアプリケーションをコンテナ化された環境で実行するための手順について説明します。

6.コンテナを使用してビルドするようにアプリケーションを変換する

アプリケーションをコンテナで実行するための変換を行う前に、コンテナを使用してアプリケーションをビルドする必要があります。ただし、引き続きコンテナの外部でアプリケーションを実行します。本セクションの手順を問題なく実行できるとわかっている場合は、「7. アプリケーションをテストする」の手順と合わせて実行し、コンテナでアプリケーションをビルドおよびテストすることもできます。

6.1.64 ビット リージョンを作成する

COBOL Server および Enterprise Server でサポートされている Linux コンテナ オペレーティング システムは 64 ビットのみです。したがって、移行先のプラットフォームが Linux であり、アプリケーションがこれまで 32 ビット モードで実行されていた場合は、アプリケーションおよびそのリージョンを 64 ビットに変換する必要があります。選択したプラットフォームが Windows であり、32 ビット モードがまだサポートされる場合は、必要に応じてアプリケーションを 32 ビット モードで引き続き実行できます。

本書では、64 ビットの Linux コンテナを使用することを前提としており、32 ビットから 64 ビットへの切り替えに必要な手順を示しています。

32 ビット リージョンから 64 ビット リージョンへの切り替えが必要な場合、最も簡単な方法は、エクスポートした XML リージョン定義を編集し、「mfCAS64Bit」要素を検索して値を 1 に設定することです。または、Enterprise Server Administration を使用して 32 ビット リージョンのコピーを作成し、[Working Mode] オプションを使用して、新しいコピーが 64 ビットであると指定することもできます。この方法でリージョンをコピーすると、ビット体系を除いて、古いリージョンと新しいリージョンが同じように構成されます。

アプリケーションで使用するリージョンを 32 ビット モードから 64 ビット モードに切り替える場合は、アプリケーションを再コンパイル、リビルド、および再テストする必要もあります。これらの作業については、コンテナ化に固有のものではないため、本書では取り上げません。詳細については、「*64 ビット ネイティブ COBOL プログラミング*」を参照してください。

*6.2.*ビルド コンテナを作成する

「*2.1. ビルド環境およびランタイム環境を定義する*」で作成したドキュメントの情報を使用して、まず、アプリケーションのビルドに必要なすべてのソフトウェアを含むコンテナ イメージを作成する必要があります。

アプリケーションでサード パーティ製ソフトウェアを使用していない場合、この手順で必要となるのは、Enterprise Developer に付属するコンテナ デモンストレーションで **bld.sh** または **bld.bat** を実行して、Docker コンテナ用の Enterprise Developer ビルド ツールを作成することだけです。詳細については、「*Enterprise Developer ベース イメージのコンテナ デモンストレーションの実行*」を参照してください。

アプリケーションに追加のソフトウェアが必要な場合は、そのソフトウェアを含むコンテナ イメージも作成する必要があります。このイメージを生成するために使用するプロセスは自動化できる必要があり、スクリプト、バッチ ファイル、およびその他の使用ファイル (Dockerfile など) はバージョン管理に保存される必要があります。

*6.3.*アプリケーションをビルドする

これで、前のセクションで準備したコンテナを使用して、コンテナへのソース コードのボリューム マウントを行い、アプリケーションをビルドできるようになりました。

ビルドを実行する一般的なコマンドは次のとおりです。

```
docker run -rm -v /src:/home/myapp -w /src microfocus/entdevhub:
rhel7_6.0_x64_login ant -lib /opt/microfocus/VisualCOBOL/lib/mfant.jar -
f .cobolBuild -logger com.microfocus.ant.CommandLineLogger
```

または

```
docker run -rm -v /src:/home/myapp -w /src microfocus/entdevhub:
rhel7_6.0_x64_login /src/buildmyapp.sh
```

ここでは、次の Docker オプションを使用します。

- `-rm`：コンテナの終了時にコンテナのファイル システムを自動的に削除することを指定します。
- `-v`：コンテナにマウントするボリュームを指定します。
- `-w`：コンテナ内でバイナリを実行するためのデフォルトの作業ディレクトリを指定します。

アプリケーションが 1 つ以上の IMTK サービスで構成されている場合は、ビルド プロセスの一環として、それらのサービスの COBOL アーカイブ (**.car**) ファイルを作成する必要があります

(後でコマンド ライン ユーティリティ `mfdepinst` を使用して、これらのファイルをディプロイします)。詳細については、「*mfdepinst* コマンドを使用してディプロイ パッケージをインストールするには」および「*mfdepinst* コマンド」を参照してください。Visual Studio および Eclipse プロジェクトには、プロジェクト ビルドの一環としてこのパッケージ化を自動的に実行するオプションが用意されています。

7.アプリケーションをテストする

コンテナを使用してアプリケーションをビルドした後、リビルドしたアプリケーションをテストして、エラーが生じていないことを確認してください。

8.アプリケーションを変換してコンテナ イメージに組み込む

このプロセスの次の手順は、アプリケーションのバイナリ ファイルおよびアプリケーションの実行に必要な構成ファイルを含むコンテナ イメージの作成です。本セクションでは、これを行うための手順について説明します。

コンテナ自体の内容が作成後に変わることはありませんが、環境変数またはボリューム マウントされたファイルおよびディレクトリを使用して、実行時にコンテナ イメージを構成できます。これにより、資格情報「シークレット」、トランスポート層セキュリティ (TLS) 証明書/キー、データベース接続文字列などの項目を構成できます。

8.1.Dockerfile を作成する

コンテナ イメージを作成する最も簡単な方法は、`docker` や `podman` など、プラットフォームのコンテナ ビルド ユーティリティで Dockerfile を使用することです。

Dockerfile には、アプリケーションをコンテナにビルドするために必要なすべての命令が含まれています。Micro Focus では、複数のステージから成る Dockerfile の使用をお勧めしています。これにより、コンパイラなどの関連ユーティリティを含むコンテナを使用してアプリケーションをビルドできますが、本番アプリケーション コンテナはこれらのユーティリティなしで実行されるため、セキュリティ上のリスクが軽減されます。詳細については、Docker Web サイトの「*Use multi-stage builds*」を参照してください。

複数のステージから成る Dockerfile を使用すると、アプリケーションを 1 つのステージでビルドし、後のステージで最終的なディプロイ イメージを構築できます。継続的インテグレーション (CI) パイプラインなど、同じことを実現するために使用できる方法は他にもありますが、複数のステージから成る Dockerfile を使用すれば、通常はほとんどのプラットフォームへの移植性を得られます。

> **注：**Docker を使用する場合、複数のステージから成るビルドを利用するには、Docker 17.05 以降を使用する必要があります。

Visual COBOL/Enterprise Developer プロジェクトを使用して IDE からアプリケーションをビルドする場合は、Visual COBOL/Enterprise Developer でテンプレート Dockerfile を自動的に作成できます。Eclipse では、アプリケーション エクスプローラー ビュー、COBOL エクスプローラー ビュー、またはプロジェクト エクスプローラー ビューでプロジェクトを右クリックし、[**New > Dockerfile**] をクリックします。Visual Studio では、ソリューション エクスプローラーでプロジェクトを右クリックし、[**Add > COBOL Docker Support**] をクリックします。詳細については、「ネイティブ *COBOL* プロジェクトに *Dockerfile* を追加するには」を参照してください。

必ず、Dockerfile をバージョン管理システムに追加してください。

8.1.1. コンテナ内でビルドする

コンテナ内でビルドするには、Dockerfile で、最初にすべてのソース コードを「ビルド コンテ

キスト」からコンテナにコピーし、次にコンテナ内から必要なツールを呼び出してアプリケーションをビルドする必要があります。

すでにコンテナを使用してアプリケーションをビルドしているため、コンテナ内でビルドするようにこのプロセスを変換するのは簡単です。また、適切なベース コンテナおよびそのコンテナ内で実行するコマンド ラインも作成済みです。

次の Dockerfile コマンドは、どのようにこれが行われるかの例を示しています。

```
# Build the application
FROM microfocus/entdevhub:sles15.1_6.0_x64 as BuildBankDemo
COPY Sources /Sources/
COPY Projects /Projects/
RUN . $MFPRODBASE/bin/cobsetenv $MFPRODBASE && \
    COBCPY=$COBCPY:$COBDIR/src/enterpriseserver/exit && \
       cd /Projects/Eclipse/BankDemo && \
       $COBDIR/remotedev/ant/apache-ant-1.9.9/bin/ant -lib
$COBDIR/lib/mfant.jar  -logger com.microfocus.ant.CommandLineLogger -
f .cobolBuild imscobbuild
```

8.1.2.単体テストをビルドおよび実行する

以前に作成した Micro Focus 単体テストがある場合は、それを Dockerfile 内のステージとしてビルドおよび実行できます。詳細については、「*Micro Focus Unit Testing Framework*」を参照してください。

テストのビルドはアプリケーションのビルドと似たプロセスであり、Dockerfile で、最初にすべてのテスト ソース コードを「ビルド コンテキスト」からコンテナにコピーし、次にコンテナ内から必要なツールを呼び出してテストをビルドする必要があります。

Dockerfile の個別のステージで、COPY 文によって前のステージから必要なファイル (アプリケーションおよびテストのバイナリ モジュールを含む) をコピーし、次に `cobmfurun` コマンドによってテストを実行します (以下を参照)。

```
# Build the MFUnit tests
FROM microfocus/entdevhub:sles15.1_6.0_x64 as BuildUnitTests
COPY Sources /Sources/
COPY Projects /Projects/
RUN . $MFPRODBASE/bin/cobsetenv $MFPRODBASE && \
    cd /Projects/Eclipse/BankDemoUnitTests && \
       $COBDIR/remotedev/ant/apache-ant-1.9.9/bin/ant -lib
$COBDIR/lib/mfant.jar -logger com.microfocus.ant.CommandLineLogger -
f .cobolBuild -DpathVar.sources=../../../Sources


# Run the MFUnit tests
FROM microfocus/entdevhub:sles15.1_6.0_x64 as RunUnitTests
RUN mkdir /runtests
COPY --from=BuildBankDemo /Projects/Eclipse/BankDemo/deploy/*.so /runtests/
COPY --from=BuildUnitTests
/Projects/Eclipse/BankDemoUnitTests/New_Configuration.bin/BankDemoUnitTests.*
/runtests/
RUN . $MFPRODBASE/bin/cobsetenv $MFPRODBASE && \
    cd runtests && cobmfurun64 -jenkins-ci BankDemoUnitTests.so
```

8.1.3.ディプロイ コンテナ イメージを構築する

1 つのステージでアプリケーションをビルドした後、Dockerfile の別のステージを使用して、ディプロイ コンテナ イメージを構築します。ディプロイ コンテナ イメージは、サード パーティ

製ソフトウェア (データベース ドライバーなど) を追加して拡張された適切な Enterprise Server イメージに基づいている必要があります。Dockerfile の 1 つのステージでこの「ベース」イメージを作成してから、後のステージにおいて、前のステージでビルドしたアプリケーション プログラムのバイナリ モジュールおよびディプロイ パッケージ (さらにその他の必要なファイル) を、ビルド コンテキストからのすべての構成ファイルとともに、イメージにコピーする必要があります。これらのファイルは、アプリケーションで要求される場所にコピーして必要な権限を設定してください。

次の Dockerfile コマンドは、どのようにこれが行われるかの例を示しています。

```
# Create the base image with needed dependencies
FROM microfocus/entserver:sles15.1_6.0_x64 as base

# Install rsyslog to get "logger" command
RUN zypper install -y hostname wget curl rsyslog

FROM base as publish
# Create directories
RUN mkdir -p /home/esadm/deploy/Logs /home/esadm/deploy/RDEF
/home/esadm/ctflogs
# Copy application binary files
COPY --from=BuildBankDemo /Projects/Eclipse/BankDemo/deploy/*.so
/home/esadm/deploy/loadlib/
# Copy BMS mod binary files
COPY Projects/Eclipse/BankDemo/loadlib/*.MOD /home/esadm/deploy/loadlib/
# COPY XA Switch module
COPY --from=BuildXASwitch /xa/*.so /home/esadm/deploy/loadlib/

# Copy catalog - expected to be deployed into mfdbfh
COPY System/catalog /home/esadm/deploy/catalog
# Copy scripts used to start and stop the container
COPY System/startserver.sh /home/esadm/deploy/
COPY System/dbfhdeploy.sh /home/esadm/deploy/
COPY System/vaultconfig.sh /home/esadm/deploy/
COPY System/pre-stop.sh /home/esadm/deploy/
# Copy the region configuration
COPY System/bankdemo.xml /home/esadm/deploy/
COPY System/bankdemo_grps.xml /home/esadm/deploy/
COPY System/bankdemo _sit.xml /home/esadm/deploy/
COPY System/ bankdemo_stul.xml /home/esadm/deploy/
# Change permissions of copied files prior to switching from root
RUN chmod +x /home/esadm/deploy/dbfhdeploy.sh && \
    chmod +x /home/esadm/deploy/vaultconfig.sh && \
    chmod +x /home/esadm/deploy/startserver.sh && \
    chmod +x /home/esadm/deploy/pre-stop.sh && \
    chown -R esadm /home/esadm && \
    chgrp -R esadm /home/esadm
```

Dockerfile は、前の手順で特定したシステム環境変数も設定する必要があります。

```
# Set the system environment variable referenced as the root directory
# within the region configuration
ENV ESP /home/esadm/deploy
# Application expects LANG=C to ensure character encoding is correct
ENV LANG C
# Ensure credentials are accessed from the secrets vault
ENV MFDS_USE_VAULT Y
```

詳細については、Docker Web サイトの「*Dockerfile reference*」で ENV 命令に関する説明を参照してください。

イメージをビルドする際には、Enterprise Server の構成、つまり MFDS およびリージョン構成もインポートする必要があります。これにより、コンテナの起動時にこのインポートを実行する必要がなくなります。ビルド プロセスでディプロイ パッケージ (**.car** ファイル) に組み込まれた IMTK サービスは、`mfdepinst` コマンド ライン ユーティリティを使用して Enterprise Server にインストールできます。

これは次のように実行できます。

```
# Import the region definition into the MFDS directory
RUN /bin/sh -c '. $MFPRODBASE/bin/cobsetenv && \
       /bin/sh -c "$COBDIR/bin/mfds64 &" && \
       /bin/sh -c "while ! curl --output /dev/null --silent --fail
http://127.0.0.1:$CCITCP2_PORT; do sleep 1; done; "&& \
       $COBDIR/bin/mfds64 /g 5 /home/esadm/deploy/bankdemo.xml S && \
       cd /home/esadm/deploy && $COBDIR/bin/mfdepinst myservice.car
       mv /var/mfcobol/logs/journal.dat /var/mfcobol/logs/import_journal.dat'
```

アプリケーションの実行時、ユーザー名 (UID) が、アプリケーションの実行に必要な最小権限のユーザーとして設定されていることを確認してください。必要がない限り、「root」は使用しないでください。詳細については、Docker Web サイトの「*Dockerfile reference*」で USER 命令に関する説明を参照してください。

これは次のように実行できます。

```
# Swap away from being the root user so Enterprise Server is not running
# with elevated privileges
USER esadm
```

root に代わるユーザーを使用する場合、その影響として、1024 未満のネットワーク ポートにバインドするための特別な権限がそのユーザーに必要になります。デフォルトでは、MFDS ではポート 86 が使用されるため、root 以外のユーザーに対して MFDS を起動する場合は、`CCITCP2_PORT` 環境変数を設定して、MFDS がリッスンするデフォルトのポートをオーバーライドしてください。この環境変数は、Dockerfile で次のような ENV 文を使用して設定できます。

```
ENV CCITCP2_PORT 34570
```

イメージは使用するすべてのリスナー ポートを宣言する必要もあり、イメージが実行するコマンドは MFDS および Enterprise Server を起動する必要があります。

```
EXPOSE $CCITCP2_PORT 34567-34569 10086
ENTRYPOINT ["/home/esadm/deploy/startserver.sh"]
```

HEALTHCHECK 文を Dockerfile に追加して、Enterprise Server リージョンが実行されていることを確認します。

```
HEALTHCHECK --interval=30s CMD ps -eaf | grep casmgr | grep -v grep || exit 1
```

詳細については、Docker Web サイトの「*Dockerfile reference*」で HEALTHCHECK 命令に関する説明を参照してください。

ユーザー資格情報や証明書などのシークレットをコンテナ イメージに含めないでください (コンテナ レジストリに保持されている間は安全でないため)。また、イメージをリビルドせずに変更する必要があるその他のリソース (構成ファイルなど) も含めないでください。このような要素は、環境変数を使用するか、コンテナへのファイルのボリューム マウントを行うことで、実行時にコンテナに「挿入」されます。

次の `docker run` コマンドの例は、パスワードを定義する環境変数を指定し、証明書の詳細を含むフォルダーのボリューム マウントを行う方法を示しています。

```
docker run -e password=SYSAD -v /certificates:/apps/bankdemo/certificates
bankdemo
```

コンテナの起動時に実行される実行可能スクリプトを作成する必要があります。つまり、このスクリプトは、Dockerfile で ENTRYPOINT または CMD コマンドによって指定します。スクリプトでは、MFDS および Enterprise Server を起動する前に環境が設定されていることを確認する必要があります。

スクリプトの例を次に示します。

```
#!/bin/bash
. $MFPRODBASE/bin/cobsetenv $MFPRODBASE

echo Starting MFDS
$COBDIR/bin/mfds64 &
while ! curl --output /dev/null --silent --fail
http://127.0.0.1:$CCITCP2_PORT; do sleep 1; done;

echo Waiting for DB to come on-line
until $COBDIR/bin/dbfhdeploy list sql://PGSQL/JCLTEST; do sleep 2; done;

# Start the Enterprise Server, reading credentials from the vault
echo Starting the Enterprise Server
$COBDIR/bin/casstart /R$ES_SERVER /S:C /U`mfsecretsadmin read
microfocus/CAS/casstart-user` /P`mfsecretsadmin read microfocus/CAS/casstart-
password`

# Wait for console log to be created signalling the server is running
while [ ! -f /var/mfcobol/es/$ES_SERVER/console.log ]; do sleep 3; done;

# Output the console log until the ES daemon terminates
CASCD_PID=`pgrep cascd64`
tail -n+1 --pid=$CASCD_PID -F /var/mfcobol/es/$ES_SERVER/console.log
```

「*3.8.シークレットを特定して資格情報コンテナを使用する*」にあるように、アプリケーションで「コンテナ機能」が有効になっていることを確認しておく必要があります。MFDS を起動する前に、必要な資格情報コンテナ シークレットが、実行中のコンテナ内に再作成されていることを確認してください。

環境変数またはボリューム マウントを使用してシークレット値をコンテナに渡し、`mfsecretsadmin` コマンド ライン ユーティリティを使用してこのデータを資格情報コンテナ内に設定できます。詳細については、「*mfsecretsadmin ユーティリティ*」を参照してください。

たとえば、次のコマンドでは、`DB_PASSWORD` 環境変数の値を使用して **pgsql.pg.postgres.password** シークレットが資格情報コンテナに再作成されます。

```
mfsecretsadmin write microfocus/mfdbfh/pgsql.pg.postgres.password
$DB_PASSWORD
```

ESCWA、MFDS、または Enterprise Server Monitor and Control (ESMAC) を使用してサーバーをリモートで監視しているユーザーに環境変数の値が表示されるのを防ぐために、MFDS を起動する前にコンテナでこれらの環境変数を設定解除することをお勧めします。

console.log ファイルの内容は、コンテナ外から簡単に利用できるようにする必要があります。上記のスクリプト例では、そのために `tail` コマンドを使用しています。Micro Focus Communications Server (MFCS) の **log.html** など、その他のログも、監視および診断の目的に役立ちます。

詳細については、「*Enterprise Server* ログ ファイル」および「通信プロセス ログ ファイル」を参照してください。

*8.2.*コンテナで *Fileshare* を実行する

Fileshare では、コンソール入力を使用して、トレースの有効化やサーバーのシャットダウンなどの管理操作を実行します。コンテナ内で実行する場合、この機能は使用できないため、Fileshare からエラー メッセージが発行されます。この問題を避けるには、起動時に –b オプションを使用して、バックグラウンド モードで実行すること (Linux の場合)、またはサービスとして実行すること (Windows の場合) を指定する必要があります。詳細については、「バックグラウンド プロセスとしての *Fileshare* の実行」および「*Windows* サービスとしての *Fileshare* の実行」を参照してください。

管理機能は FSView を使用して実行する必要があるため、適切な FSView 資格情報を指定して資格情報コンテナを使用するように Fileshare を構成する必要があります。これを行うには、Fileshare の起動時に `/uv` オプションを使用します。

次の構成ファイルを考慮する必要があります。

- **fs.cfg**
- **fhredir.cfg**
- **dbase.ref**

詳細については、「*Fileshare* サーバーの構成オプション」を参照してください。

*8.3.*コンテナのセキュリティに関する考慮事項

アプリケーションのセキュリティ要件については、慎重に検討してください。どのディプロイ環境でも考慮する必要がある一般的なセキュリティ要件に加えて、次に示すように、コンテナで Micro Focus Server 製品を使用および構成する場合に特に考慮すべき領域が複数あります。

- 外部セキュリティ マネージャー (ESM) を、Enterprise Server へのアクセスを制御する手段として現時点で使用していない場合は、使用を開始してください。たとえば、Active Directory またはその他のセキュリティ マネージャーを使用して、Enterprise Server および ESCWA へのアクセスを制限できます。
- リスナーの使用を可能な限り制限してください。
- アプリケーションの実行に明示的に必要ではないポートは公開しないでください。
- セキュリティ マネージャー内で監査を有効にしてください。最大限のパフォーマンスを得るには、ローカル Syslog サーバーを実行し、ファイルへの書き込みまたは外部 Syslog サーバーへのイベントの転送 (あるいはその両方) を行うように構成してください。
- 必要最小限の権限を持つユーザー名 (UID) を使用してコンテナを実行してください。

詳細については、「*Enterprise Server* セキュリティ」および「*Enterprise Server* での監査」を参照してください。

*8.4.*コンテナのデータに関する考慮事項

アプリケーションによっては、次に示すように、コンテナで Micro Focus Server 製品を使用および構成する場合に特に考慮すべきデータ関連の問題があり、その一部またはすべてについて検討する必要があります。

- アプリケーションを Windows から Linux に (またはその逆に) 移動する場合は、次の点に注意してください。
    - データ ファイルのデフォルトのファイル拡張子は 2 つのプラットフォームで異なるため、ファイル ハンドラー構成ファイル (**extfh.cfg**) で IDXNAMETYPE オプションを使用した追加構成が必要になる場合があります。詳細については、「構成ファイル」および「*IDXNAMETYPE*」を参照してください。
    - プラットフォームによって、大文字と小文字の区別、パスおよびファイル名の区切り文字など、さまざまな領域で動作が異なります。ディプロイ先となるプラットフォームでこれらの領域についてアプリケーションが正しく処理できることを確認する必要があります。
- コンテナでは、SQL アクセスを可能にするために追加のソフトウェアおよび構成が必要になる場合があります。たとえば、ODBC を使用するには、次の作業が必要になります。
    - openODBC などのデータベース ドライバーをインストールします。
    - 適切な XA スイッチ モジュールを構築します。
    - ODBC 構成ファイルを構成します。

    使用するデータベースに応じて、必要となる手順は異なります。

    詳細については、「*RM スイッチ モジュールのビルド*」および「*XA 準拠リソース (XAR) の使用*」を参照してください。
- データ ファイルをどこに保存するかを検討する必要があります。コンテナを停止すると、コンテナ内のデータに加えられた変更はすべて失われます。これはテストでは便利な場合もありますが、本番環境での使用には適していないと考えられます。対処方法になり得る選択肢は次のとおりです。
    - 永続ストレージへのボリューム マウント
    - データ ボリューム
    - 外部データベース

9.アプリケーションをテストする

ここまでの手順で、アプリケーションをコンテナにビルドしました。次に、リビルドしたアプリケーションをテストして、その動作が変更されていないことを確認する必要があります。前に使用した自動テストを再利用して、これを行うことができます。`docker run` または `podman run` コマンドを使用して、環境変数およびボリューム マウントのディレクトリを必要に応じて設定し、アプリケーションを実行できるようにしてください。

次に例を示します。

```
docker run -rm -e ENVVAR1=yes -e ENVVAR2=no -v /user/data:/user/data
myapplication:latest
```

段階 C – スケールアウト環境にディプロイする

テスト済みのアプリケーションをコンテナで実行した後、通常は、プロセスの次の段階に進む必要があります。具体的には、スケールアウト環境でアプリケーションを実行して、アプリケーションの拡張性を高める方法を調査することになります。

スケールアウト環境では、アプリケーションは、1 つの大規模なリソースではなく、コンピューターなど、小規模なリソースを多数使用して処理されます。アプリケーションに対する負荷の増減に応じて、リソースを追加または削除するだけで、簡単に環境をスケールアップまたはスケールダウンできます。

Kubernetes などのコンテナ化されたスケールアウト環境では、コンテナの論理グループ レベルでのスケーリングが可能です。Kubernetes では、この機能をポッドと呼びます。同じノード (コンピューター) で多数のポッドを実行でき、Kubernetes コントローラーによってポッドを監視できるため、アプリケーションの負荷に応じてポッドの数を増減でき、障害が発生したポッドは自動的に再起動されます。

スケールアウト アプリケーションを作成する場合は、可用性を高めながら、異なるアプリケーション インスタンス間でデータの一貫性を維持することが課題になります。この問題に対処しやすいように、Enterprise Server には、Micro Focus データベース ファイル ハンドラー (MFDBFH) が用意されています。MFDBFH は、順編成ファイル、相対ファイル、および索引付き順編成ファイルを RDBMS に移行してスケーリングを強化できるようにします。また、パフォーマンス/可用性クラスター (PAC) を実行するための要件でもあります。MFDBFH は、現在、Visual COBOL ファイル アクセスではサポートされていません。

詳細については、「*Micro Focus* ネイティブ データベース ファイル処理およびエンタープライズ サーバー リージョン データベース管理」および「スケールアウト パフォーマンス/可用性クラスター」を参照してください。

10.スケールアウトに対応できるようにデータを変換する

スケールアウト環境にアプリケーションをディプロイする前に、データ アクセスを効果的にスケーリングできるように、データの保存方法およびアクセス方法について慎重に検討する必要があります。

MFDBFH を使用すると、アプリケーション構成を変更するだけで、つまりプログラムのソース コードを変更せずに、VSAM データを RDBMS に保存し、後でそのデータにアクセスできます。この点については、「*10.1. MFDBFH* を使用して *VSAM* を *RDBMS* に移行する」で詳しく説明します。

スケールアウト環境では Fileshare サーバーも使用できますが、Fileshare はこのような環境向けに特別に設計されたテクノロジーではないため、用途に適していることを確認する必要があります。特に、Fileshare のパフォーマンスおよび回復オプションがニーズに合っていることを確認してください。

データが RDBMS にすでに保存されている場合は、現在のサーバー構成がスケールアウトのディプロイに適しているかどうかを確認してください。

すべての非静的データを移行してください。この対象としては、カタログ化されたファイルおよびスプール ファイル (これらのファイルについては、「*3.*バージョン管理用にアプリケーション構成を準備する」で説明しているように、アプリケーション構成ファイルをバージョン管理システムに移動する際にハウスキーピングを実行する必要があります)、さらに FCT を介してアクセスするその他の CICS ファイルが挙げられます。最終的な本番環境への移行では、この作業を慎重に計画する必要があります。この点については、「*10.2.* カタログ ファイルおよびデータ ファイルを *RDBMS* にディプロイする」および「*10.3. CICS* リソース定義ファイルを *RDBMS* にディプロイする」で詳しく説明します。

Enterprise Server では、スケールアウト環境とパフォーマンス/可用性クラスター (PAC) を併用できます。PAC では、MFDBFH、およびオープンソースのインメモリ データ構造ストアである Redis を使用して、PAC のメンバー間でデータを共有します。Enterprise Server コンテナのインスタンスを複数ディプロイし、アプリケーション内でネイティブに SQL を使用してデータを

RDBMS に保存できます。Visual COBOL は、パフォーマンス/可用性クラスターおよび MFDBFH の使用をサポートしていません。詳細については、*Redis* の Web サイトおよび「スケールアウト パフォーマンス/可用性クラスター」を参照してください。

10.1.MFDBFH を使用して *VSAM* を *RDBMS* に移行する

このプロセスの一環として、MFDBFH を使用して VSAM ファイルの一部またはすべてを RDBMS に移動する場合は、次の点を考慮する必要があります。

- 相対パスを使用していないカタログ エントリがある場合は、相対パスを使用するように変換するか、MFDBFH での使用に適したパスとなるように変換してください。Unix 形式のファイル区切り文字 (「****」ではなく「**/**」) を使用して、Windows にも Linux にもディプロイできるようにしてください。

> **ヒント：**相対パスのみを使用するようにカタログを変換することをお勧めします。これにより、カタログを RDBMS にディプロイした場合に、すべてのエントリのパスがすぐに MFDBFH での使用に適した状態になります。
>
> 相対パスに変換しないエントリがある場合は、元のカタログのコピーを保持する必要が生じます。これは、`dbfhdeploy` ツールを使用してデータ ファイルを RDBMS にディプロイする際に、物理ファイルの場所を取得できるようにするためです。

- JCL ジョブ カードに PCDSN のオーバーライドがあるかどうかを確認してください。それには、ジョブ カードで `%PCDSN%` を探します。理想としては、カタログのルックアップを使用するようにこれらを変更する必要がありますが、MFDBFH に適したパスを指定することもできます。
- CICS を実行しており、カタログではなく FCT を使用している場合は、これらのエントリを MFDBFH の場所に変更する必要が生じます。

*10.2.*カタログ ファイルおよびデータ ファイルを *RDBMS* にディプロイする

カタログ ファイルおよびデータ ファイルを RDBMS にディプロイする場合は、次の点を考慮してください。

- カタログを使用して、ディプロイが必要なファイルのリストを取得する必要があります。
- ヘッダーのないファイルをディプロイする場合、つまり、固定ブロック順編成ファイルおよび行順編成ファイルをディプロイする場合は、形式およびレコード長に関する情報の入力が必要になります。この情報はカタログから取得できます。

> **ヒント：**本番環境では、これは 1 回限りの操作になります。テスト時および開発時では、「既知」のカタログおよびデータセットをディプロイすると、環境の再現性が向上します。

10.3.CICS リソース定義ファイルを *RDBMS* にディプロイする

CICS リソース定義ファイルを RDBMS にディプロイする場合は、次の点を考慮してください。

- CICS を処理するリージョンは、パフォーマンス/可用性クラスター (PAC) の一部にする必要があります。CICS リソース定義ファイルを MFDBFH にディプロイし、それを使用するようにリージョンを構成することをお勧めしますが、CICS リソース定義ファイルを各コンテナ イメージ内でローカルに複製することもできます。

> **ヒント：**CICS リソース定義ファイル内のリソースに静的な変更を加える場合は、バージョン管理にコミットする必要があります。

詳細については、「*Micro Focus* データベース ファイル処理のための *CICS* アプリケーションの

構成」、「*Micro Focus* データベース ファイル処理のための *CICS* リソースの構成」、および「*PAC* のベスト プラクティス」を参照してください。

11.スケールアウト環境にディプロイできるようにアプリケーションを構成する

スケールアウト環境でアプリケーションを実行する前に、次のように、追加の構成手順をいくつか実行する必要があります。

- SQL データベースに基づく MFDBFH など、スケールアウト対応のデータ ソースと連携するように、共有データ アクセスを再構成します。必要な追加の RDBMS ドライバーをインストールするように Dockerfile を更新します。

```
# Create the base image with needed dependencies
FROM microfocus/entserver:sles15.1_6.0_x64 as base
# Install updated ODBC driver required by mfdbfh
RUN zypper install -y unixODBC postgresql && \
        cd /tmp && wget
https://download.postgresql.org/pub/repos/zypp/11/suse/sles-12.4-
x86_64/repodata/repomd.xml.key && rpm --import ./repomd.xml.key && \
    zypper install -y
https://download.postgresql.org/pub/repos/zypp/11/suse/sles-12.4-
x86_64/postgresql11-libs-11.5-1PGDG.sles12.x86_64.rpm && \
    zypper install -y
https://download.postgresql.org/pub/repos/zypp/11/suse/sles-12.4-
x86_64/postgresql11-odbc-11.01.0000-1PGDG.sles12.x86_64.rpm
```

- **odbc.ini** や **odbcinst.ini** など、追加の構成を設定します。

```
ADD System/odbcinst.ini.su /etc/unixODBC/odbcinst.ini
```

- 適切な XA スイッチ モジュールを構築します。

```
# Build the Postgres XA switch module required by the application
FROM microfocus/entdevhub:sles15.1_6.0_x64 as BuildXASwitch
RUN . $MFPRODBASE/bin/cobsetenv $MFPRODBASE && \
    mkdir /xa && \
       cd /xa && \
       cp $COBDIR/src/enterpriseserver/xa/* . && \
       /bin/bash ./build pg
```

- MFDBFH で使用するデータベースを指定するように Enterprise Server を構成します。これには、特定のデータベース インスタンスに適した設定を使用した「XA リソース構成」の構成が含まれます。

  XA スイッチ モジュールを構成内で環境変数として指定し、その環境変数の値を Dockerfile 内で設定することも、XA スイッチ モジュールが配置されている場所のフル パスをコンテナ イメージ ファイル システム内で指定することもできます。

  詳細については、「*RM* スイッチ モジュールのビルド」および「*XA* 準拠リソース *(XAR)* の使用」を参照してください。

詳細については、「データベース ファイル処理環境変数」、「*Micro Focus* ネイティブ データベース ファイル処理およびエンタープライズ サーバー リージョン データベース管理」、および「スケールアウト パフォーマンス/可用性クラスター」を参照してください。

12.スケールアウト環境に保持されているデータを使用して、コンテナにビルドしたアプリケーションをテストする

MFDBFH などを使用してデータを別のテクノロジーに移行した場合は、元のデータにアクセスせずにアプリケーションを再テストして、エラーが生じていないこと、およびすべてのデータが正しく移行されていることを確認してください。

13.アプリケーション コンテナをスケールアウト環境にディプロイする

本セクションでは、アプリケーションをスケールアウト環境にディプロイする準備が完了した後に実行する必要のある手順について説明します。

*13.1.*スケールアウトのディプロイ用にアプリケーションを準備する
コンテナ化されたアプリケーションをスケールアウト環境にディプロイする前に、ディプロイメント記述子 (Kubernetes の **.yaml** ファイルなど) を作成し、スケールアウト アーキテクチャのすべての要素が適切に配置されていることを確認する必要があります。

本セクションでは、Kubernetes クラスターの使用を前提としています。

どのユーザー ID をアプリケーションの実行に使用するか、どのネットワーク ポートを公開するか、どのようなファイアウォール ルールを適用するかなど、アプリケーションのセキュリティ要件を慎重に検討する必要があります。これらの側面について検討する際に常に目標となるのは、セキュリティ上の脆弱性を低減するために、必要最小限の権限および最小限のオープン ポートのみを使用するということです。セキュリティ要件の実装時に作成または変更するファイルはすべてバージョン管理に保存される必要があります。

ライフサイクル フックを使用して、アプリケーションのリソースをスケールダウンしても Enterprise Server の作業が予期せず終了しないようにする必要があります。これは、長時間実行されるバッチ ジョブの場合に特に重要となります。ライフサイクル フックを使用すると、新しい作業がリージョンに割り当てられるのを防止できるとともに、現在のワークロードが完了するまでリージョンの実行が確実に維持されます。定義する自動スケーリング ルールでは、このシャットダウン手順の実行が許可される猶予期間の指定が必要です。この期間を過ぎると、ホスト インスタンスが終了します。リージョンでバッチ ジョブが実行されている場合、この猶予期間は、バッチ ジョブが完了するまでの予想所要時間よりも長くする必要があります。

アプリケーションのディプロイに使用できる有効な構成は多数あります。また、クラウド プロバイダーが管理するサービス (Redis 用 Amazon ElastiCache、Microsoft Azure Cache for Redis、Microsoft Azure Database for PostgreSQL、Amazon Aurora など) を使用することも、クラスター内またはオンプレミスで同等のサービスを実行することもできます。どの選択肢が要件に最適かを評価する必要があります。

13.1.1.Kubernetes の YAML ファイルを作成する

スケールアウト アーキテクチャのさまざまな側面に関する計画が完了した後、前の段階で作成したコンテナ イメージの Kubernetes StatefulSet を作成できます。

Enterprise Server を使用している場合は、StatefulSet 構成で、パフォーマンス/可用性クラスター (PAC) の使用について指定します。指定する項目は次のとおりです。

- PAC の名前。`ES_PAC` 環境変数を使用して指定します。
- スケールアウト リポジトリ (SOR) の構成。これには、次の環境変数の設定が含まれます。
    - `ES_DB_FH`：データベース ファイル ハンドラーのサポートを有効にします。
    - `ES_DB_SERVER`：データベース名を指定します。
    - `ES_SCALE_OUT_REPOS_1`：使用する SOR を指定します。

- `MFDBFH_CONFIG`：MFDBFH 構成を指定します。

詳細については、「*PAC および SOR の環境変数*」および「データベース ファイル処理環境変数」を参照してください。

Visual COBOL for SOA を使用している場合は、アプリケーションのラベルを指定します。ESCWA の Kubernetes 自動検出メカニズムでは、このラベルを使用して、適切なポッドを効率的に選択できます (詳細については、以下を参照してください)。Micro Focus Directory Server で使用するポート、つまり `CCITCP2_PORT` 環境変数の値を定義します。デフォルトのポート (86) が使用されていない場合でも、IANA 登録名「mfcobol」を使ってポッドで実行することにより、ディレクトリ サーバーとの通信に使用する必要のあるポートを ESCWA で自動的に検出できます。

> **ヒント：**Kubernetes シークレットは、環境変数およびファイルを使用して資格情報と証明書をアプリケーションに挿入するために使用されます。init コンテナを使用して、環境変数に格納されている資格情報を資格情報コンテナに入力してから、共有「メモリ」の emptyDir ボリュームを使用して、**/opt/microfocus/EnterpriseDeveloper/etc/secrets/microfocus** などの場所へのボリュームマウントを行うことで、これらの情報をアプリケーション コンテナに追加します。資格情報の環境変数をアプリケーション コンテナに直接挿入すると、実行中のポッドへの ESCWA アクセス権を持つすべてのユーザーにその情報が (暗号化されずに) 表示されます。
>
> 詳細については、Kubernetes Web サイトの「*Distribute Credentials Securely Using Secrets*」を参照してください。

次の YAML フラグメントは、スケールアウト機能を指定し、アプリケーションで使用できるよう PAC および SOR を構成して、アプリケーションが正しく実行されていることを確認するための活性プローブおよび準備プローブを指定する方法を示しています。

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: ed60-bnkd-statefulset-mfdbfh
  labels:
    app: ed60-bnkd-mfdbfh
spec:
  replicas: 1
  selector:
    matchLabels:
      app: ed60-bnkd-mfdbfh
  serviceName: ed60-bnkd-svc-mfdbfh
  template:
    metadata:
      labels:
        app: ed60-bnkd-mfdbfh
    spec:
      # Allow time for clean shutdown of Enterprise Server
      terminationGracePeriodSeconds: 120
      nodeSelector:
        "beta.kubernetes.io/os": linux
      securityContext:
        runAsUser: 500
        fsGroup: 500
```

```
      initContainers:
        # Initialize the local vault with needed secrets
      - name: vault-config
        image: bankdemo:latest
        imagePullPolicy: Always
        command: ["/home/esadm/deploy/vaultconfig.sh"]
        env:
        # XA Open string prefix, used with username and password in
        # vaultconfig.sh
        - name: MFDBFHOPENSTRING
          value: "DSN=PG.JCLTEST,LOCALTX=T"
        # Database connection credentials
        - name: DB_USERNAME
          valueFrom:
            secretKeyRef:
              name: pgsql-secret
              key: db-username
        - name: DB_PASSWORD
          valueFrom:
            secretKeyRef:
              name: pgsql-secret
              key: db-password
        # OpenLDAP connection password
        - name: LDAP_PASSWORD
          valueFrom:
            secretKeyRef:
              name: ldap-secret
              key: ldap-password
        # Credentials used to start region
        - name: ES_PASSWORD
          valueFrom:
            secretKeyRef:
              name: es-secret
              key: es-password
        - name: ES_USERNAME
          valueFrom:
            secretKeyRef:
              name: es-secret
              key: es-username
        volumeMounts:
        - name: vault-volume
          mountPath:
/opt/microfocus/EnterpriseDeveloper/etc/secrets/microfocus
        # Shared region workarea
        - name: region-workarea-volume
          mountPath: /var/mfcobol/es/BANKDEMO
      containers:
        # Main application container - Enterprise Server running the
        # application
      - name: application
        image: bankdemo:latest
        imagePullPolicy: Always
        env:
        # Performance and Availability Cluster identifier
        - name: ES_PAC
          value: "MYPAC"
        # Enable use of MFDBFH
        - name: ES_DB_FH
```

```
          value: "true"
        # Set MFDBFH configuration
        - name: MFDBFH_CONFIG
          value: "/home/esadm/MFDBFH.cfg"
        # Set name of MFDBFH server Enterprise Server should use - must
        # match server name in MFDBFH.cfg
        - name: ES_DB_SERVER
          value: "PGSQL"
        # Primary Scale Out Repository configuration
        - name: ES_SCALE_OUT_REPOS_1
          value: "RedisLocal=redis,ed60-bnkd-svc-redis:6379##TMP#TD=*#TS=*"
        # Force MFDS to use secrets vault
        - name: MFDS_USE_VAULT
          value: "Y"
        # Location of trusted root certificate for OpenLDAP server
        - name: OPENLDAP_CAROOT
          value: /var/run/secrets/microfocus/ca.crt
        # Port opened by sidecar running syslog daemon
        - name: SYSLOG_PORT
          value: "2514"
        ports:
        - containerPort:  34568
          name: mfcobol-ws
          protocol: TCP
        - containerPort:  34570
          name: mfcobol
          protocol: TCP
        - containerPort:  34571
          name: telnet
          protocol: TCP
        lifecycle:
          preStop:
            exec:
              command: ["/home/esadm/deploy/pre-stop.sh"]
        readinessProbe:
          httpGet:
            path: /esmac/casrdo00
            port: 34568
          initialDelaySeconds: 5
          periodSeconds: 10
        livenessProbe:
          httpGet:
            path: /esmac/casrdo00
            port: 34568
          initialDelaySeconds: 60
          periodSeconds: 30
        volumeMounts:
        # Vault initialized by init-container
        - name: vault-volume
          mountPath:
/opt/microfocus/EnterpriseDeveloper/etc/secrets/microfocus
        # Database configuration
        - name: iniconfig-volume
          mountPath: /etc/unixODBC/odbc.ini
          subPath: odbc.ini
        # MFDBFH configuration
        - name: mfdbfh-config-volume
          mountPath: /home/esadm/MFDBFH.cfg
```

```
          subPath: MFDBFH.cfg
        # Shared region workarea - sidecars process log files from this
        # location
        - name: region-workarea-volume
          mountPath: /var/mfcobol/es/BANKDEMO
```

上記のフラグメントには、**vaultconfig.sh** ファイルおよび **pre-stop.sh** ファイルへの参照が含まれています。**vaultconfig.sh** の内容の例を次に示します。

```
#!/bin/bash

. $MFPRODBASE/bin/cobsetenv

echo Setting up the vault
# Setup the XA connection string in the vault
$COBDIR/bin/mfsecretsadmin write
microfocus/MFDS/1.2.840.5043.07.001.1573035249.139659451564033-OpenString
$MFDBFHOPENSTRING,USRPASS=$DB_USERNAME.$DB_PASSWORD
# Setup the MFDBFH password secrets
$COBDIR/bin/mfsecretsadmin write
microfocus/mfdbfh/pgsql.pg.cas.crossregion.password $DB_PASSWORD
$COBDIR/bin/mfsecretsadmin write microfocus/mfdbfh/pgsql.pg.cas.mypac.password
$DB_PASSWORD
$COBDIR/bin/mfsecretsadmin write microfocus/mfdbfh/pgsql.pg.datastore.password
$DB_PASSWORD
$COBDIR/bin/mfsecretsadmin write microfocus/mfdbfh/pgsql.pg.jcltest.password
$DB_PASSWORD
$COBDIR/bin/mfsecretsadmin write microfocus/mfdbfh/pgsql.pg.postgres.password
$DB_PASSWORD
$COBDIR/bin/mfsecretsadmin write microfocus/mfdbfh/pgsql.pg.utilities.password
$DB_PASSWORD
# Setup ESM LDAP password
$COBDIR/bin/mfsecretsadmin write
microfocus/MFDS/ESM/1.2.840.5043.14.001.1573468236.2-LDAPPwd $LDAP_PASSWORD
# Setup ES admin credentials
$COBDIR/bin/mfsecretsadmin write microfocus/CAS/casstart-user $ES_USERNAME
$COBDIR/bin/mfsecretsadmin write microfocus/CAS/casstart-password $ES_PASSWORD
```

pre-stop.sh の内容の例を次に示します。

```
#!/bin/bash

# Request the region is stopped
. $MFPRODBASE/bin/cobsetenv && casstop -r$ES_SERVER -u`mfsecretsadmin read
microfocus/CAS/casstart-user` -p`mfsecretsadmin read microfocus/CAS/casstart-
password`

# Loop until the server has stopped
while [ ! -f /var/mfcobol/es/BANKDEMO/shutdown.txt ]; do sleep 3; done;
```

次の YAML フラグメントは、サイドカー コンテナを使用して、診断目的に必要なログ ファイルを Kubernetes ロギング フレームワークに出力する方法を示しています。

```
      # Sidecar for logging mfcs log.html
       - name: mfcs-log
         image: bankdemo:latest
```

```
        command: ["/bin/bash", "-c", "while [ ! -f
/var/mfcobol/es/BANKDEMO/log.html ]; do sleep 3; done; tail -n+1 -F
/var/mfcobol/es/BANKDEMO/log.html"]
        lifecycle:
          preStop:
            exec:
              # Wait for the server to signal to have been shutdown then
              # stop outputting the mfcs log
              command: ["/bin/bash", "-c", "while [ ! -f
/var/mfcobol/es/BANKDEMO/shutdown.txt ]; do sleep 3; done; TAIL_PID=`pgrep
tail`; kill -s SIGTERM $TAIL_PID"]
        volumeMounts:
        - name: region-workarea-volume
          mountPath: /var/mfcobol/es/BANKDEMO
```

次の YAML フラグメントは、サイドカーを使用してポッド内で Syslog デーモンを実行し、Enterprise Server の監査出力を受信して、リモートの Syslog デーモンに転送する方法を示しています。

```
- name: mfaudit-log
        image: bankdemo:latest
        command: ["rsyslogd", "-n", "-f", "/etc/mfaudit/rsyslog.conf"]
        imagePullPolicy: "Always"
        ports:
        - name: incoming-logs
          containerPort: 2514
        lifecycle:
          preStop:
            exec:
              # Wait for the server to signal to have been shutdown then
              # terminate the syslog daemon
              command: ["/bin/sh", "-c", "while [ ! -f
/var/mfcobol/es/BANKDEMO/shutdown.txt ]; do sleep 3; done; RSYSLOG_PID=`pgrep
rsyslogd`; kill -s SIGTERM $RSYSLOG_PID"]
        volumeMounts:
        # RSYSLOG Configuration - rsyslog.conf loaded from configmap
        - name: syslog-conf-volume
          mountPath: /etc/mfaudit/
        - name: region-workarea-volume
          mountPath: /var/mfcobol/es/BANKDEMO
```

次の YAML フラグメントは、サイドカーを使用して Prometheus メトリックス プロバイダーを実行し、Enterprise Server メトリックスに基づいて水平ポッド自動スケーリングでスケーリングできるようにする方法を示しています。

通常は、3270 や Web サービス リスナーなど、公開する必要のあるアプリケーション リスナーに対して Kubernetes サービスを定義し、それらの前にロード バランサーを配置します。次の YAML はこれを行う方法を示しています。

```
kind: Service
apiVersion: v1
metadata:
  name: ed60-bnkd-svc-mfdbfh
spec:
  selector:
    app: ed60-bnkd-mfdbfh
```

```
  ports:
    # Web Service listener port - also used by readiness/liveness probes
    - name: mfcobol-ws
      protocol: TCP
      port: 34568
      targetPort: 34568
    # Directory server listener port
    - name: mfcobol
      protocol: TCP
      port: 86
      targetPort: 34570
---
kind: Service
apiVersion: v1
metadata:
  name: ed60-bnkd-svc-mfdbfh-tn
spec:
  selector:
    app: ed60-bnkd-mfdbfh
  ports:
    # Telnet listener port
    - name: telnet
      protocol: TCP
      port: 23
      targetPort: 34571
  type: LoadBalancer
```

13.1.1.1.クラスターを作成する (または既存のクラスターにアクセスする)

スケールアウト クラスターを作成します (Amazon Elastic Kubernetes Service (Amazon EKS) や Azure Kubernetes Service (AKS) などを使用)。また、クラスターおよびその機能について把握する必要があります。

13.1.1.2.Redis およびデータベース サーバーが利用可能であることを確認する

パフォーマンス/可用性クラスター (PAC) で Enterprise Server を実行している場合は、実行中の Redis サーバーを使用していることを確認する必要があります。アプリケーション用に適切な Redis サーバーをディプロイするか、クラウド プロバイダーの適切な Redis サービス (AWS ElastiCache、Azure Redis Cache、Google Cloud Memorystore など) を利用します。

選択したデータベース サーバーが利用可能である必要もあります。Kubernetes サービスを作成して Redis およびデータベース サーバーに接続をルーティングし、Enterprise Server リソースの構成時にこれらのサービス アドレスを使用します。次に例を示します。

```
apiVersion: v1
kind: Service
metadata:
  labels:
    app: ed60-bnkd-svc-redis
  name: ed60-bnkd-svc-redis
spec:
  externalName: <address of Redis server>
  selector:
    app: ed60-bnkd-svc-redis
  type: ExternalName
status:
  loadBalancer: {}
---
```

```
apiVersion: v1
kind: Service
metadata:
  labels:
    app: pgsql-svc
  name: ed60-bnkd-svc-pgsql
spec:
  externalName: <address of database server>
  selector:
    app: pgsql-svc
  type: ExternalName
status:
  loadBalancer: {}
```

13.1.1.3.ESCWA をディプロイする

アプリケーションを簡単に監視できるようにするには、ESCWA インスタンスをクラスターにディプロイする必要があります。PAC (Enterprise Server にディプロイしている場合) または Kubernetes の動的検出 (COBOL Server にディプロイしている場合) を監視するように、ESCWA インスタンスを構成します。

Kubernetes クラスター内で Visual COBOL for SOA を使用している場合は、同じ Kubernetes クラスター内で ESCWA を実行し、Enterprise Server ポッドのクラスターをスキャンするように構成できます。

詳細については、「*Kubernetes での ESCWA の使用*」を参照してください。

たとえば、サイドカー コンテナを使用し、ポッドのポート 8001 を使って kubectl プロキシを実行し、コマンド ラインで ESCWA を開始します。

```
--K8sConfig={"Direct":false,
             "Port":"8001",
             "LabelEnvironment":false,
             "Label":"app%3Dmyapp-label"}
```

Enterprise Server を使用しており、パフォーマンス/可用性クラスター (PAC) の一部としてサーバーを実行している場合は、代わりに、または追加で ESCWA を構成して、パフォーマンス/可用性クラスターのメンバーを表示できます。

たとえば、ESCWA を開始する際には次のオプションを使用します。

```
--SorList=[{"SorName":"MySorName",
            "SorDescription":"My PAC instance",
            "SorType":"redis",
            "SorConnectPath":"my-redis-server:6379",
            "Uid" : "1"}]
```

ESCWA は、TLS および ESM セキュリティを有効にして構成する必要があります。また、ESCWA のディプロイ環境を複製することは現在サポートされていないため実行できません。詳細については、「*ESCWA のトランスポート層セキュリティ (TLS)*」および「外部セキュリティ マネージャーの指定」を参照してください。

14.スケールアウトでアプリケーション コンテナのレプリカを複数実行してアプリケーションをテストおよび調整する

アプリケーションをスケールアウト環境にディプロイした後、アプリケーションをもう一度テストして調整する必要があります。

パフォーマンスと回復力の最適なバランスを実現するには、レプリカの数および各レプリカ内の SEP の数を調整する必要があります。LoadRunner や UFT などのツールを使用して、さまざまなシナリオをテストできます。詳細については、「*LoadRunner Professional*」および「*UFT One*」を参照してください。

15.メトリックスおよび自動スケーリングを使用する

Kubernetes は、さまざまなアプリケーションの要求に対応できるようにポッドの自動スケーリングをサポートしています。この機能は Enterprise Server で使用できますが、制限があることに注意してください。

Kubernetes の自動スケーリングはステートレス REST アプリケーションなどのステートレス アプリケーションで特に効果を発揮しますが、Enterprise Server アプリケーションの多くは、CICS 3270 アプリケーションによく見られるように、本質的にステートフルです。3270 ユーザー セッションは特定のサーバー ポッド インスタンスに接続されると、そのポッドに「固定された状態」になります。つまり、ポッドが過負荷になった場合に、クラスターをスケールアップしても既存のセッションのパフォーマンスが向上するとは限りません。また、クラスターをスケールダウンすると、そのスケールダウンの一環として終了されたポッドに接続していたアクティブ セッションは切断されるため、実行中であったトランザクションは完了しない場合があります。

アプリケーションにどのような負荷があるかは把握しているものの、日中のオンライン負荷や夜間のバッチ負荷など、時間帯によって負荷が異なる場合は、Kubernetes の自動スケーリングを使用するよりも、アプリケーションを手動 (または時間指定) でスケーリングした方が適切なことがあります。次の形式のコマンドを使用すると、これを実現できます。

```
kubectl scale --replicas=NN statefulset/statefulsetname)
```

Kubernetes の水平ポッド自動スケーリングは、`HorizontalPodAutoscaler` 定義によって構成します。この定義では、スケーリングを行う範囲となるレプリカの最小数および最大数、スケーリングの基礎として使用するメトリックスの名前 (およびそのターゲット値) などの詳細を指定します。

Kubernetes には、メモリ使用率や CPU 使用率など、各種のメトリックスが組み込まれていますが、これらが常に Enterprise Server での使用について最適なメトリックスというわけではありません。カスタムの Prometheus メトリックスによって標準の Kubernetes メトリックスを補完できます。また、Enterprise Server には `casuetst` というイベント マネージャー出口モジュールが用意されており、この出口モジュールが環境変数 `ES_EMP_EXIT_n` によって有効化されている場合 (`ES_EMP_EXIT_1=casuetst` など)、**ESmonitor1.csv** というファイルが作成されます。これは単純なカンマ区切り値ファイルで、1 分あたりに処理されたタスクの数、平均タスク待ち時間、平均タスク存続時間、タスク キューの長さなどの情報を含みます。

具体的には、**ESmonitor1.csv** ファイルの行は次の形式になります。

```
YYYYMMDDhhmmssnn,Tasks-PM,AvLatency,AvTaskLen,-Queue--,TotalTasks,SEPcount,-
Dumps--,FreeSMem,
```

詳細は次のとおりです。

YYYYMMDDhhmmssnn	メトリックスが記録された日時
Tasks-PM	1 分あたりに処理されたタスクの数
AvLatency	平均タスク待ち時間
AvTaskLen	平均タスク存続時間
-Queue--	キューに登録されたトランザクション数
TotalTasks	システムで実行されたタスクの総数
SEPcount	使用中の SEP の数
-Dumps--	Enterprise Server によって取得されたダンプの数
FreeSMem	空き共有メモリの量

これらのメトリックスは Prometheus サーバーに追加でき、その情報を問い合わせるように Kubernetes メトリックス サーバーを構成できます (Kubernetes Prometheus アダプターを使用)。

サイドカー コンテナを使用し、**ESmonitor1.csv** ファイルの内容を読み取って関連する値を Prometheus メトリックス形式で公開する小さなプログラムを実行することにより、Enterprise Server メトリックスを公開できます。これを簡単に行えるように、さまざまなプログラミング言語のクライアント ライブラリが用意されています。最も使いやすいのは、Golang バージョンです。

Golang プログラムのサンプルを次に示します。この例では、Enterprise Server メトリックス用の Prometheus「ゲージ」を作成し (ゲージは、バックグラウンド スレッドで **ESmonitor1.csv** ファイルの値の変化に基づいて最新の状態に保たれます)、ポッドのポート 8080/metrics に対する http `GET` 要求によって返されたメトリックス自体を公開します。

```
package main

import (
    "io/ioutil"
    "log"
    "net/http"
    "os"
    "strconv"
    "strings"
    "time"
    "github.com/prometheus/client_golang/prometheus"
    "github.com/prometheus/client_golang/prometheus/promhttp"
)

func recordMetrics() {
    // Start background thread which reads ESmonitor1.csv and
    // updates the gauges, then sleeps 30 seconds and repeats
    go func() {
        arg1 := os.Args[1]
        time.Sleep(60 * time.Second)
        for {
            data, err1 := ioutil.ReadFile(arg1)
            if err1 != nil {
                log.Printf("File reading error: %v", err1)
                time.Sleep(60 * time.Second)
            }
            log.Printf("Contents of file: %s", string(data))

            s := strings.Split(string(data), ",")
            i1, err := strconv.ParseFloat(s[1], 64)
            log.Printf("Tasks per minute : %f", i1)
            i2, err := strconv.ParseFloat(s[2], 64)
```

```
            log.Printf("Average Latency : %f", i2)
            i3, err := strconv.ParseFloat(s[3], 64)
            log.Printf("Average task length : %f", i3)
            i4, err := strconv.ParseFloat(s[4], 64)
            log.Printf("Queued tasks : %f", i4)
            if err != nil {
                log.Printf("convert to float error: %v", err)
            }
            tPM.Set(i1)
            avgLatency.Set(i2)
            avgTaskDuration.Set(i3)
            workQueued.Set(i4)
            time.Sleep(30 * time.Second)
        }
    }()
}

// Create the gauges
var (
    tPM = prometheus.NewGauge(prometheus.GaugeOpts{
        Name: "es_tasks_per_minute",
        Help: "number of tasks per minute",
    })
    avgLatency = prometheus.NewGauge(prometheus.GaugeOpts{
        Name: "es_average_task_latency",
        Help: "average latency",
    })
    workQueued = prometheus.NewGauge(prometheus.GaugeOpts{
        Name: "es_queued_transactions",
        Help: "amount of work queued",
    })
    avgTaskDuration = prometheus.NewGauge(prometheus.GaugeOpts{
        Name: "es_average_task_duration",
        Help: "average task duration",
    })
)

func init() {
    // Metrics have to be registered to be exposed:
    prometheus.MustRegister(tPM)
    prometheus.MustRegister(avgLatency)
    prometheus.MustRegister(avgTaskDuration)
    prometheus.MustRegister(workQueued)
}

func main() {

    recordMetrics()

    // The Handler function provides a default handler to expose metrics
    // via an HTTP server. "/metrics" is the usual endpoint for that.
    http.Handle("/metrics", promhttp.Handler())

    port := os.Getenv("LISTENING_PORT")

    if port == "" {
        port = "8080"
    }
```

```
    log.Printf("listening on port:%s", port)

    err := http.ListenAndServe(":"+port, nil)
    if err != nil {
        log.Fatalf("Failed to start server:%v", err)
    }
}
```

サイドカーの定義例を次に示します。

```
- name: custom-metrics
  image: es-metrics:latest
  imagePullPolicy: "Always"
  ports:
  - name: metrics
    containerPort: 8080
  volumeMounts:
  - name: region-workarea-volume
    mountPath: /esmetric/workarea
```

Kubernetes の自動スケーリングは Kubernetes Metrics Server への問い合わせを行うことで機能するため、Prometheus メトリックスを自動スケーリングに使用するには、まず Kubernetes Metrics Server からメトリックスにアクセスできる必要があります。これを実現するには、必要な Prometheus メトリックスの詳細を示す構成を指定して `k8s-prometheus-adapter` を使用し、アプリケーション ポッド テンプレート仕様に注釈を追加して、Prometheus がポッドから実際のメトリックス値を取得するようにします。

詳細については、Kubernetes Web サイトの「*Metrics Server*」、GitHub の「*k8s-prometheus-adapter*」、および GitHub の「*Prometheus*」の「*Scraping Pod Metrics via Annotations*」セクションを参照してください。

たとえば、次のポッドの注釈は、Prometheus がポッドのポート 8080:/metrics URL からメトリックスを取得することを示しています。

```
      metadata:
        annotations:
          prometheus.io/scrape: "true"
          prometheus.io/path: /metrics
          prometheus.io/port: "8080"
```

Prometheus アダプターは、GitHub の「*Configuration Walkthroughs*」で説明しているように、Kubernetes Metrics Server に追加するメトリックス値の詳細を示すルールを使用して構成する必要があります。

次の例は、これを行う方法を示しています。

```
rules:
  default: true
  custom:
    - seriesQuery: 'es_tasks_per_minute'
      seriesFilters: []
      resources:
        overrides:
          kubernetes_namespace:
```

```
            resource: namespace
          kubernetes_pod_name:
            resource: pod
      name:
        matches: "es_tasks_per_minute"
        as: ""
      metricsQuery: <<.Series>>{<<.LabelMatchers>>,container_name!="POD"}
```

次に示すように、適切な `HorizontalPodAutoscaler` リソースを適用することで、Kubernetes 水平ポッド オートスケーラーでメトリックスを使用できます。

```
apiVersion: autoscaling/v2beta1
kind: HorizontalPodAutoscaler
metadata:
  name: hpa-bankdemo
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: StatefulSet
    name: ed60-bnkd-statefulset-mfdbfh
  minReplicas: 1
  maxReplicas: 5
  metrics:
  - type: Pods
    pods:
      metricName: es_tasks_per_minute
      targetAverageValue: 100
```