Visual COBOL チュートリアル

COBOL 開発: コンテナを利用した SOA 開発

1. 目的

コンテナ技術は、Linux カーネルのコンテナ機能を使って実行環境を他のプロセスから隔離し、その中でアプリケーションを動作させることがで きます。また、コンテナプロセスの起動に必要なシステム資源は、仮想マシンの起動と比較すると非常に軽量です。コンテナ技術の利用により、 アプリケーションとライブラリを同一のコンテナ内に固められるため、容易にアプリケーションの移動やディプロイが行えます。 Visual COBOL は、コンテナ技術として Docker、もしくは Podmanを利用することができます。 コンテナ技術を利用することにより COBOL 開発に以下の利点を提供します。

- 開発・実行環境をイメージで保持するため、CI ツールとの連携による日々の自動テストや回帰テストの実施や、同一環境の複数立ち 上げが非常に容易
- バージョン毎にイメージが作成されるため、パッチアップデートを含めたバージョンアップ検証作業において複数環境構築が不要

本チュートリアルでは、コンテナ環境内で SOA アプリケーションを稼働させ、外部からのアクセスを確認します。

2. 前提

- 本チュートリアルで使用したマシン OS : Red Hat Enterprise Linux 8.6
- Linux 環境に Red Hat のサブスクリプション情報が登録済みであること
- Visual COBOL 9.0 Development Hub コンテナ製品をご購入のお客様
- コンテナコマンドの知識があること
- 別チュートリアル「ステップバイステップチュートリアル コンテナを利用した開発」を実施済みであること

本チュートリアルでは、一部の手順において、下記リンク先のサンプルファイルを使用します。事前にダウンロードをお願いします。 サンプルプログラムのダウンロード

3. チュートリアルの流れ

本章では、製品コンテナイメージを利用した開発に必要な操作や機能を以下の順に紹介します。

- 1. コンテナ内で開放したポートへのアクセス
- 2. コンテナ環境内で SOA アプリケーションの稼働

Red Hat Enterprise Linux 7.x, 8.x, 9.x いずれの環境をご利用のお客様も本チュートリアルは実施できますが、説明時に コンテナコマンドの違いを吸収するため、以下の用語を使用します。

<コンテナコマンド>: 7.x 環境のお客様は docker, 8.x, 9.x 環境のお客様は podman に読み替えてください。

また、本チュートリアルでは、コンテナイメージは RHEL 8.6 上で Visual COBOL 9.0J の PatchUpdate02 を使用しています。異なるバージョンをご利用のお客様は、適宜、イメージ名を変更してください。

3.1. コンテナ内で開放したポートへのアクセス

コンテナ環境を起動すると、ホスト環境上に構築された独自のネットワークが構成されます。このため、ホスト環境外からはアドレス 解決ができず、アクセスができません。これを解決するためにコンテナ起動時に、ホスト環境上に指定したポートを開放し、本ポート を介してホスト環境外からのアクセスを可能としています。

コンテナ内で稼働する COBOL 専用アプリケーションサーバーの管理コンソール画面は、通常製品同様、デフォルトではポート 10086 でリッスン状態となります。こちらを利用して、コンテナ内へのアクセス方法を確認します。

3.1.1. コンテナアドレスを指定したアクセス

1) 以下のコマンドを利用して、コンテナ環境を起動します。

<コンテナコマンド> run --rm -itd --name test microfocus/vcdevhub:rhel8.6_9.0_x64_pu02

podman run --rm -itd --name test microfocus/vcdevhub:rhel8.6_9.0_x64_pu02
0ebef183c3e5c967eec542fbd3e136e16c50593c779595ff488530dd94a2fdaf

2) 以下のコマンドを利用して、COBOL 専用アプリケーションサーバーをコンテナ内で起動します。

<コンテナコマンド> exec -d test sh -c ". ¥\$MFPRODBASE/bin/cobsetenv && escwa --

BasicConfig.MfRequestedEndpoint=tcp:*:10086"

podman exec -d test sh -c ". ¥\$MFPRODBASE/bin/cobsetenv && escwa --BasicConfig.MfRequestedEndpoint=tcp:*:10086"

5beff6553c973409f90133b006bdf8993d8c15328f29038a2280459f88354aa3

補足)

管理コンソール画面は、デフォルトでは localhost 以外からのアクセスを拒否するモードで稼働します。これを外部からアク セス可能とするために、引数 "BasicConfig.MfRequestedEndpoint"を追加しています。

3) inspect コマンドを利用して、コンテナの IP アドレスを確認します。

<コンテナコマンド> inspect test |grep IPAddress

podman inspect test |grep IPAddress

"IPAddress": "**10.88.0.161**",

4) curl コマンドを利用して、管理コンソールへのアクセスを確認します。

curl http://10.88.0.161:10086 -o /dev/null -w 'STAUS CODE=%{http_code}¥n'



5) localhost:86 でアクセスできないことを確認します。

curl -I -X GET http://localhost:10086 -o /dev/null -sS

curl http://localhost:10086 -o /dev/null -sS

curl: (7) Failed to connect to localhost port 10086: 接続を拒否されました

6) コンテナ環境を終了・破棄します。

<コンテナコマンド> stop test

podman stop test test

3.1.2. publish オプションを利用したアクセス

先の例では、コンテナに割り当てられたアドレスを指定した場合に限り、アクセスできています。このコンテナのアドレスを参照できない 環境、例えば、ホスト環境外からのアクセスはできないことを意味しています。ここでは、publish オプションを利用してホスト環境 側のポートを介してコンテナ内へ要求を転送する方法を確認します。

1) 以下のコマンドを利用して、コンテナ環境を起動します。

以下は1行で実行してください。

<コンテナコマンド> run --rm -itd -p 10086:10086 --name test

microfocus/vcdevhub:rhel8.6_9.0_x64_pu02

podman run --rm -itd -p 10086:10086 --name test microfocus/vcdevhub:rhel8.6_9.0_x64_pu02 de94747742c05e0deb18bc3bc104b5c878f30b4f97307999d05021d79f05112e

2) 以下のコマンドを利用して、COBOL 専用アプリケーションサーバーをコンテナ内で起動します。

<コンテナコマンド> exec -d test sh -c ". ¥\$MFPRODBASE/bin/cobsetenv && escwa --

BasicConfig.MfRequestedEndpoint=tcp:*:10086"

podman exec -d test sh -c ". ¥\$MFPRODBASE/bin/cobsetenv && mfds" 5beff6553c973409f90133b006bdf8993d8c15328f29038a2280459f88354aa3 3) curl コマンドを利用して、管理コンソールへのアクセスを確認します。

curl http://localhost:10086 -o /dev/null -w 'STAUS CODE=%{http_code}¥n'



先の例では、localhost へのアクセスは拒否されていましたが、今回はアクセスできています。

```
これは、publish オプション (-p) により、ホスト環境上のポート 10086 への要求がコンテナ内のポート 10086 に転送されているため、localhost からのアクセスが可能になっています。
```

補足)

ここでは、最初の 10086 がホスト環境側のポートを指定しています。例えば、ホスト環境のポート 40086 を介して、コンテナ側のポート 10086 に要求を転送する場合は -p 40086:10086 と指定します。

4) コンテナ環境を終了・破棄します。

<コンテナコマンド> stop test

podman stop test

test

3.1.3. EXPOSE 命令を利用したアクセス

EXPOSE 命令は、コンテナイメージを作成する Containerfile (RHEL7.x の docker 環境では Dockerfile) 内に記述します。もしくは、コンテナ起動時に --expose オプションで都度指定することもできます。

EXPOSE によるポート指定は、publish オプションのようなポートの開放は行われません。コンテナ環境を起動する際に、

publish オプションを利用して、指定したホスト環境のポートと紐づけを行うか、publish all オプション (-P) を指定する必要があります。

それでは、EXPOSE についての挙動を確認します。

3.1.3.1. **EXPOSE のみの動作**

1) EXPOSE 命令を含めたコンテナイメージを新たに作成します。

任意のディレクトリにて、Containerfile (RHEL7.x 環境をご利用のお客様は Dockerfile) を以下の内容で作成しま

す。

```
FROM microfocus/vcdevhub:rhel8.6_9.0_x64_pu02
```

EXPOSE 10086 注意)

1 行目は、ご利用のコンテナイメージ・タグ名に変更してください。

2) 以下のコマンドを利用して、新たなコンテナイメージを作成します。

<コンテナコマンド> build -t test .

	# podman build -t test .
	STEP 1/2: FROM microfocus/vcdevhub:rhel8.6_9.0_x64_pu02
	STEP 2/2: EXPOSE 10086
	COMMIT test
	> 688fb12e8cf
	Successfully tagged localhost/test:latest
	688fb12e8cf13398cf034587efdc42ee9f8ad4901943c9eb8ef0621e4abe822a
	test という名前のコンテナイメージが作成されます。
3)	オプションを指定せず、コンテナ環境を起動します。
	<コンテナコマンド> runrm -itdname test test
	# podman runrm -itdname test test
	27ef2bbd0b11f84acdff47a08fb9a87eca4e701750b15ead2f4ec697b64fe10d
5)	以下のコマンドを利用して、管理画面をコンテナ内で起動します。
	<コンテナコマンド> exec -d test sh -c ". ¥\$MFPRODBASE/bin/cobsetenv && escwa
	BasicConfig.MfRequestedEndpoint=tcp:*:10086"
	# podman exec -d test sh -c ". ¥\$MFPRODBASE/bin/cobsetenv && escwa
	BasicConfig.MfRequestedEndpoint=tcp:*:10086"
	ad822ea4c2a18af7d22f13b959e33bc27d05b95b8d74f96b48f466523a53b5b0
6)	コンテナアドレスを指定した場合、管理コンソール画面にアクセスできることを確認します。
	<コンテナコマンド> inspect test grep IPAddress
	# podman inspect test grep IPAddress

"IPAddress": "10.88.0.169",

curl http://10.88.0.169:10086 -o /dev/null -w 'STAUS CODE=%{http_code}¥n'

curl http://10.88.0.169:10086 -o /dev/null -w 'STAUS CODE=%{http_code}*n'
% Total % Received % Xferd Average Speed Time Time Time Current
Dload Upload Total Spent Left Speed
100 5419 100 5419 0 0 352k 0 --:--:- --:-- 377k
STAUS CODE=200

7) localhost でのアクセスができないことを確認します。

curl http://localhost:10086 -o /dev/null -sS

curl http://localhost:10086 -o /dev/null -sS

curl: (7) Failed to connect to localhost port 10086: 接続を拒否されました

8) 現在のコンテナ環境を停止・破棄します。

<コンテナコマンド> stop test

podman stop test test

3.1.3.2. **publish all オプションを指定した動作**

1) publish all オプション (-P) を指定して、コンテナ環境を起動します。

<コンテナコマンド> run --rm -itd -P --name test test

podman run --rm -itd -P --name test test

dfee1042f84b34d68338fe64035979eeb9cfa70a1c47a741cc0eeacc255d1892

このオプションは、publish オプション (-p) と異なり、自動的にホスト環境上で利用可能なポートと EXPOSE 指定された ポートを紐づけます。この紐づけは、以下のコマンドで確認できます。

<コンテナコマンド> ps

	# podman ps
	CONTAINER ID IMAGE COMMAND CREATED STATUS
	PORTS NAMES
	4166003564a6 localhost/test:latest /bin/bash 58 seconds ago Up 57 seconds ago
	0.0.0:36021->10086/tcp test
	上記では、PORTS 項目にて、ホスト環境のポート 36021 がコンテナ環境のポート 10086 に紐づけられていることが分
	かります。
	補足)
	inspect コマンドを用いても、ポートの紐づけ情報を確認できます。
2)	また、publish オノション (-p) と併用することで、手動でホートの紐つけもできます。
2)	以下のコマンドを利用して、COBOL 専用アフリケーションサーハーをコンテナ内で起動します。
	<コンテナコマンド> exec -d test sh -c ". ¥\$MFPRODBASE/bin/cobsetenv && escwa
	BasicConfig.MfRequestedEndpoint=tcp:*:10086"
	# podman exec -d test sh -c ". ¥\$MFPRODBASE/bin/cobsetenv && escwa
	BasicConfig.MfRequestedEndpoint=tcp:*:10086"
	15d8d7963d08cfd67f2c20ccd2953c64bbcb95bafb8a1b63e8b99a050f19dbbb
3)	localhost に対してアクセスが行えることを確認します。
	curl http://localhost:36021 -o /dev/null -w 'STAUS CODE=%{http_code}¥n'
	36021 は、上記で確認したポート番号に修正してください。
	# curl http://localhost:36021 -o /dev/null -w 'STAUS CODE=%{http_code}¥n'
	% Total % Received % Xferd Average Speed Time Time Time Current
	Dload Upload Total Spent Left Speed
	100 5419 100 5419 0 0 251k 0::: 251k
	STAUS CODE=200
4)	コンテナ環境を停止・破棄します。
	<コンテナコマンド> stop test
	# podman stop test
	toot

5) テスト用に作成したコンテナイメージを削除します。

<コンテナコマンド> rmi test

podman rmi test

Untagged: localhost/test:latest

Deleted: d277e965d21aded6a62c3ebf2ab66704ed7618f82e947125dc8c8cda11ccf6be

3.2. コンテナ環境内で SOA アプリケーションの稼働

本節では、サンプル SOA アプリケーションである書籍情報管理アプリケーションをコンテナ内で稼働させ、正常に動作することを確認していきます。本節で使用するサンプルアプリケーションは、以下の構成となります。



書籍情報管理アプリケーションは、索引ファイルを使用して、書籍情報の新規追加、参照、削除の機能を有しています。各機能 は、RESTful ウェブサービスとして実装されています。ホスト環境のポート 39002 にリクエストを送信すると、コンテナ内で稼働す る COBOL アプリケーションが実行され、レスポンスとして、それぞれの処理結果が JSON 形式で戻されます。 以下に、各機能のエンドポイント URL と、サンプルとなるリクエストデータ例を示します。

サービス名	API 例
書籍情報検索	http://localhost:39002/temppath/BookRest/1.0/SearchBook
	リクエストデータ例)
	{
	"LNK_B_STOCKNO": "1111"
	}

サービス名	API 例
書籍情報追加	http://localhost:39002/temppath/BookRest/1.0/AddBook
	リクエストデータ例
	{
	"LNK_B_DETAILS":
	{
	"LNK_B_TEXT_DETAILS":
	{
	"LNK_B_TITLE": "Alice's Adventures in Wonderland",
	"LNK_B_TYPE": "Fantasy",
	"LNK_B_AUTHOR": "Lewis Carroll",
	},
	"LNK_B_STOCKNO": 9999,
	"LNK_B_RETAIL": 100,
	"LNK_B_ONHAND": 200,
	"LNK_B_SOLD": 300,
	}
	}
書籍情報削除	http://localhost:39002/temppath/BookRest/1.0/DeleteBook
	リクエストデータ例)
	{
	"LNK_B_STOCKNO": "9999"
	}

1) サンプルファイルを任意のディレクトリに解凍し、vc-tutorial02 ディレクトリ配下に移動します。

サンプルファイルには、以下のリソースが含まれています。

リソース名	説明
BOOKINFO.dat	書籍情報を管理する索引ファイル
	バインドマウントにより、コンテナ内から参照される。
DEMOSV.xml	サンプルアプリケーションを稼働させるためのアプリケーションサーバーインスタン
	スの定義ファイル
	setup.sh 内でインポート処理が行われる。
deploy/	サンプルアプリケーションのアーカイブファイル
	setup.sh 内でアプリケーションサーバーに登録される。
req/*	RESTful ウェブサービス実行時のリクエストデータ例
script/setup_internal.sh	コンテナイメージを作成する際に、コンテナ内部で実行されるセットアップ内容
	を記載したスクリプト
setup.sh	サンプルアプリケーションを含むコンテナを構築するためのセットアップスクリプト

- setup.sh 内の1行目に記載されているコンテナイメージ名を、ご利用の環境に合わせて修正、保存してください。
 VC_PROD_IMAGE=microfocus/vcdevhub:rhel8.6_9.0_x64_pu02
 あわせて、2行目に記載されている CONTAINER_CMD を必要に応じて更新してください。
- 3) コンテナイメージを作成するため、以下のコマンドを実行します。

sh setup.sh

sh setup.sh 853ba602385afe13db0c85661275554751b861c7ed62693b827f2496957f30c6 (中略) Writing manifest to image destination Storing signatures 8cb46d6ea47f17cc096a13c02b3ad8a03ea75f15958041d6e351435df541f622 mfcs_wk mfcs_bookrest イメージが作成されていることを確認します。

<コンテナコマンド> images

# podman images		
REPOSITORY	TAG	IMAGE ID
CREATED SIZE		
localhost/mfcs_bookrest	latest	8cb46d6ea47f 45
seconds ago 1.03 GB		

4) 以下のコマンドで、コンテナ環境を起動します。

以下は1行で実行してください。

<コンテナコマンド> run --rm -tid --name demo -v \$PWD:/var/data:z -p 40086:10086 -p 39002:9002 mfcs bookrest

podman run --rm -tid --name demo -v \$PWD:/var/data:z -p 40086:10086 -p 39002:9002 mfcs_bookrest

a3f3177d2a8f34204e52afb6649df970960aca56b6073d095dea9fca18f55e37

5) 以下のコマンドで、コンテナ内で管理画面を起動します。

<コンテナコマンド> exec -d demo sh -c ". ¥\$MFPRODBASE/bin/cobsetenv && escwa --

BasicConfig.MfRequestedEndpoint=tcp:*:10086"

podman exec -d demo sh -c ". ¥\$MFPRODBASE/bin/cobsetenv && escwa --

BasicConfig.MfRequestedEndpoint=tcp:*:10086"

cdf3847bb3e63cff2705d382822f5788b4f8bde37569c5c718a50b80800d7ad1

6) 以下のコマンドで、コンテナ内で COBOL 専用のアプリケーションサーバーを起動します。

<コンテナコマンド> exec -d demo sh -c ". ¥\$MFPRODBASE/bin/cobsetenv && mfds"

podman exec -d demo sh -c ". ¥\$MFPRODBASE/bin/cobsetenv && mfds"

8b511d71ccbc7794fcef1b728973ce4cec08b366c106b0e4815cf993961c1575

7) 以下のコマンドで、コンテナ内でサンプルアプリケーションが登録されたインスタンスを起動します。

<コンテナコマンド> exec -d demo sh -c ". ¥\$MFPRODBASE/bin/cobsetenv && casstart /rDEMOSV"

podman exec -d demo sh -c ". ¥\$MFPRODBASE/bin/cobsetenv && casstart /rDEMOSV" ca57dd99d724463ad6667cc55034b9c7bafdbbadbfe035c409c1565958b79856

8) 管理コンソール画面へのアクセスが行えることを確認します。

```
curl http://localhost:40086 -o /dev/null -w 'STAUS CODE=%{http_code}¥n'
# curl http://localhost:40086 -o /dev/null -w 'STAUS CODE=%{http_code}¥n'
% Total % Received % Xferd Average Speed Time Time Time Current
Dload Upload Total Spent Left Speed
100 5419 100 5419 0 0 793k 0 --:--:-- --:-- 881k
STAUS CODE=200
```

9) リクエストデータ例で紹介したリクエストを実行し、正常に処理が行われていることを確認します。

```
書籍情報検索)
```

curl -X POST http://localhost:39002/temppath/BookRest/1.0/SearchBook -d @req/SearchBook

```
#curl -X POST http://localhost:39002/temppath/BookRest/1.0/SearchBook -d
@req/SearchBook
{
    "LNK_B_DETAILS" :
    {
        "LNK_B_TEXT_DETAILS" :
        {
            "LNK_B_TITLE" : "LORD OF THE RINGS",
            "LNK_B_TYPE" : "FANTASY",
            "LNK_B_AUTHOR" : "TOLKIEN"
        },
        "LNK_B_STOCKNO" : "1111",
        "LNK_B_RETAIL" : 1500,
        "LNK_B_ONHAND" : 4000,
        "LNK_B_SOLD" : 3444
    },
        "LNK_FILE_STATUS" : "00"
```

書籍情報追加)

curl -X POST http://localhost:39002/temppath/BookRest/1.0/AddBook -d @req/AddBook

curl -X POST http://localhost:39002/temppath/BookRest/1.0/AddBook -d @req/AddBook

"LNK_FILE_STATUS" : "00"

書籍情報削除)

curl -X POST http://localhost:39002/temppath/BookRest/1.0/DeleteBook -d @req/DeleteBook

curl -X POST http://localhost:39002/temppath/BookRest/1.0/DeleteBook -d @req/DeleteBook

"LNK_FILE_STATUS" : "00"

補足)

本例では、localhost:39002 という形でアクセスしていますが、ホスト環境外からのアクセスすることもできます。アクセスできない場合は、ファイアウォール設定などをご確認ください。

10) コンテナ環境を停止・破棄します。

<コンテナコマンド> stop demo

podman stop demo

demo

11) コンテナイメージを削除します。

<コンテナコマンド> rmi mfcs_bookrest

podman rmi mfcs_bookrest

Untagged: localhost/mfcs_bookrest:latest

Deleted: dfaf2e950bd54bad0bbecd0f56dd9985ff1c93633ffb8f21d4906edbea7ecd58

4. 補足

4.1. サンプルスクリプトについて

説明のため、スクリプト内には存在しない行番号を設定しています。

4.1.1. setup.sh

1: VC_PROD_IMAGE=microfocus/vcdevhub:rhel8.6_9.0_x64_pu02 2: CONTAINER_CMD=podman 3: \$CONTAINER_CMD run -itd -v \$PWD:/var/data:z --rm --name mfcs_wk \$VC_PROD_IMAGE 4: \$CONTAINER_CMD exec -d mfcs_wk sh -c ". ¥\$MFPRODBASE/bin/cobsetenv && mfds" 5: sleep 5 6: \$CONTAINER_CMD exec mfcs_wk sh -c "sh /var/data/script/setup_internal.sh" 7: \$CONTAINER_CMD commit mfcs_wk mfcs_bookrest 8: \$CONTAINER_CMD stop mfcs_wk

行数	説明
3~4	1 行目に指定した Visual COBOL 製品コンテナイメージからコンテナ環境を起動し、コンテナ内部で
	COBOL 専用のアプリケーションサーバーを起動しています。
6	script/setup_internal.sh を利用して、コンテナ内でセットアップ処理を行っています。
7~8	セットアップが完了したコンテナ環境を mfcs_bookrest というコンテナイメージ名として保存しています。その
	後、8 行目にてコンテナ環境の停止・破棄を行っています。

4.1.2. script/setup_internal.sh

1:#!/bin/sh
2: . \$MFPRODBASE/bin/cobsetenv
3: mfds /g 5 /var/data/DEMOSV.xml O
4: cd /var/data/deploy && mfdepinst BookRest.car
5: mfds /s 1
6: while :
7: do
8: PROCINF=`ps -efa grep mfds grep -v grep`
9: if ["\$PROCINF" == ""]; then
10: break
11: fi
12: sleep 1
13: done
14: exit 0

行数	説明
2	Visual COBOL 製品を使用するために必要な環境変数を設定しています。
3~5	サンプルで使用するアプリケーションサーバーインスタンス DEMOSV の定義をインポートしたうえで、アプリケー
	ションファイル BookRest.car のディプロイを行います。その後、アプリケーションサーバーを停止します。
6~13	アプリケーションサーバーの完全停止を確認しています。

WHAT'S NEXT

• 本チュートリアルで学習した技術の詳細については製品マニュアルをご参照ください。

免責事項

ここで紹介したソースコードは、機能説明のためのサンプルであり、製品の一部ではございません。ソースコードが実際に動作するか、御社業務に適合するかなどに関しまして、一切の保証はございません。 ソースコード、説明、その他すべてについて、無謬性は保障されません。 ここで紹介するソースコードの一部、もしくは全部について、弊社に断りなく、御社の内部に組み込み、そのままご利用頂いても構いません。 本ソースコードの一部もしくは全部を二次的著作物に対して引用する場合、著作権法の精神に基づき、適切な扱いを行ってください。