

## 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 : Amazon Linux 2023
- Visual COBOL 10.0 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、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](#)

## 内容

- 1 目的
- 2 前提
- 3 チュートリアルの流れ
  - 3.1 コンテナ内で開放したポートへのアクセス
    - 3.1.1 コンテナアドレスを指定したアクセス
    - 3.1.2 publish オプションを利用したアクセス
    - 3.1.3 EXPOSE 命令を利用したアクセス
  - 3.2 コンテナ環境内で SOA アプリケーションの稼働
- 4 補足
  - 4.1 サンプルスクリプトについて
    - 4.1.1 setup.sh
    - 4.1.2 script/setup\_internal.sh

### 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

稼働環境によって異なるコンテナ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>: docker または、podman 動作環境でサポートしているコマンド

また、本チュートリアルでは、コンテナイメージは Amazon Linux 2023 上で Visual COBOL 10.0 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:amzn2023\_10.0\_x64

docker run --rm -itd --name test microfocus/vcdevhub:amzn2023\_10.0\_x64

```
[ec2-user@al2023-v-na ~]$ docker run --rm -itd --name test microfocus/vcdevhub:amzn2023_10.0_x64
30349514fe696b338e9a16713dbbc06cba6e32ac78d7075c75e31bb9b17e6d8a
[ec2-user@al2023-v-na ~]$
```

- 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"

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

```
[ec2-user@al2023-v-na ~]$ docker exec -d test sh -c ". ¥$MFPRODBASE/bin/cobsetenv && escwa --BasicConfig.MfRequestedEndpoint=tcp:*:10086"
[ec2-user@al2023-v-na ~]$
```

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 |grep -v Secondary  
docker inspect test |grep IPAddress |grep -v Secondary

```
[ec2-user@al2023-v-na ~]$ docker inspect test |grep IPAddress |grep -v Secondary
      "IPAddress": "172.17.0.2",
      "IPAddress": "172.17.0.2",
[ec2-user@al2023-v-na ~]$
```

- 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://172.17.0.2:10086 -o /dev/null -w 'STAUS CODE=%{http\_code}¥n'

```
[ec2-user@al2023-v-na ~]$ curl http://172.17.0.2:10086 -o /dev/null -w 'STAUS COD
E=%{http_code}¥n'
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload  Total  Spent    Left  Speed
100 3096 100 3096    0     0 71789      0 --:--:-- --:--:-- --:--:-- 70363
STAUS CODE=200
[ec2-user@al2023-v-na ~]$
```

ステータスコード 200 が戻されていることから、アクセスできたことを確認してください。

- 5) localhost:10086 でアクセスできないことを確認します。

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

```
[ec2-user@al2023-v-na ~]$ curl -I XGET http://localhost:10086 -o /dev/null -sS
curl: (6) Could not resolve host: XGET
curl: (7) Failed to connect to localhost port 10086 after 0 ms: Couldn't connect to se
rver
[ec2-user@al2023-v-na ~]$
```

- 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

docker stop test

```
[ec2-user@al2023-v-na ~]$ docker stop test
test
[ec2-user@al2023-v-na ~]$
```

### 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:amzn2023_10.0_x64
```

```
[ec2-user@al2023-v-na ~]$ docker run --rm -itd -p 10086:10086 --name test ¥
microfocus/vcdevhub:amzn2023_10.0_x64
2b5db941c9e769e323be6920f3554078199420503c18fd455db3ca9c380402c7
[ec2-user@al2023-v-na ~]$
```

- 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"
```

```
[ec2-user@al2023-v-na ~]$ docker exec -d test sh -c ". ¥$MFPRODBASE/bin/cobsetenv
&& escwa --BasicConfig.MfRequestedEndpoint=tcp:*.10086"
[ec2-user@al2023-v-na ~]$
```

- 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'
```

```
[ec2-user@al2023-v-na ~]$ curl http://localhost: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 3096 100 3096 0 0 143k 0 --:--:-- --:--:-- --:--:-- 143k
STAUS CODE=200
[ec2-user@al2023-v-na ~]$
```

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
```

```
docker stop test
```

```
[ec2-user@al2023-v-na ~]$ docker stop test
test
[ec2-user@al2023-v-na ~]$
```

### 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 (docker サポート環境では 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。

#### 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 (Amazon Linux 2023 などの docker サポート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:amzn2023_10.0_x64
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 .`

`docker build -t test .`

```
[ec2-user@al2023-v-na ~]$ docker build -t test .
[+] Building 0.1s (5/5) FINISHED                                docker:default
=> [internal] load build definition from Dockerfile              0.0s
=> => transferring dockerfile: 152B                               0.0s
=> [internal] load metadata for docker.io/microfocus/vcdevhub:amzn2023_10.0_x64
                                                                0.0s
=> [internal] load .dockerignore                                  0.0s
=> => transferring context: 2B                                     0.0s
=> CACHED [1/1] FROM docker.io/microfocus/vcdevhub:amzn2023_10.0_x64
                                                                0.0s
=> exporting to image                                             0.0s
=> => exporting layers                                           0.0s
=> => writing image sha256:fdfea68572f45e4a4e1df9452951f98f91137d78de122f3a793dd93e42813973
                                                                0.0s
=> => naming to docker.io/library/test                           0.0s
[ec2-user@al2023-v-na ~]$
```

`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> run --rm -itd --name test test

docker run --rm -itd --name test test

```
[ec2-user@al2023-v-na ~]$ docker run --rm -itd --name test test
f24161fbc15ae284c6a598742091d34f7d87b9c5deae7999e1babbc46db2dd91
[ec2-user@al2023-v-na ~]$
```

- 4) 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"

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

```
[ec2-user@al2023-v-na ~]$ docker exec -d test sh -c ". ¥$MFPRODBASE/bin/cobsetenv && escwa -- BasicConfig.MfRequestedEndpoint=tcp:*:10086" ^C
[ec2-user@al2023-v-na ~]$ docker exec -d test sh -c ". ¥$MFPRODBASE/bin/cobsetenv && escwa -- BasicConfig.MfRequestedEndpoint=tcp:*:10086"
[ec2-user@al2023-v-na ~]$
```

- 5) 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 |grep -v Secondary

```
[ec2-user@al2023-v-na ~]$ docker inspect test |grep IPAddress |grep -v Secondary
      "IPAddress": "172.17.0.2",
      "IPAddress": "172.17.0.2",
[ec2-user@al2023-v-na ~]$
```

- 6) 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

```
[ec2-user@al2023-v-na ~]$ curl http://localhost:10086 -o /dev/null -sS
curl: (7) Failed to connect to localhost port 10086 after 0 ms: Couldn't connect to server
[ec2-user@al2023-v-na ~]$
```

- 7) 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

docker stop test

```
[ec2-user@al2023-v-na ~]$ docker stop test
test
[ec2-user@al2023-v-na ~]$
```

### 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

```
[ec2-user@al2023-v-na ~]$ docker run --rm -itd -P --name test test
2ed1a6371217476395e3d540bd037356c6d11d7705772995f93178a21ceea749
[ec2-user@al2023-v-na ~]$
```

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

```
[ec2-user@al2023-v-na ~]$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS
RTS
2ed1a6371217   test     "/bin/bash"             26 seconds ago Up 26 seconds 0.0.0.0:32768->10086/tcp, :::32768->10086/tcp test
[ec2-user@al2023-v-na ~]$
```

上記では、PORTS 項目にて、ホスト環境のポート 32768 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。

また、publish オプション (-p) と併用することで、手動でポートの紐づけもできます。

- 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"

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

```
[ec2-user@al2023-v-na ~]$ docker exec -d test sh -c ". ¥$MFPRODBASE/bin/cobsetenv && escwa -- BasicConfig.MfRequestedEndpoint=tcp:*:10086"
[ec2-user@al2023-v-na ~]$
```

- 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:32768 -o /dev/null -w 'STAUS CODE=%{http\_code}¥n'

32768 は、上記で確認したポート番号に修正してください。

```
[ec2-user@al2023-v-na ~]$ curl http://localhost:32768 -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 3096 100 3096 0    0 184k    0 --:--:-- --:--:-- --:--:-- 188k
STAUS CODE=200
[ec2-user@al2023-v-na ~]$
```

- 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

docker stop test

```
[ec2-user@al2023-v-na ~]$ docker stop test
test
[ec2-user@al2023-v-na ~]$
```

- 5) テスト用に作成したコンテナイメージを削除します。

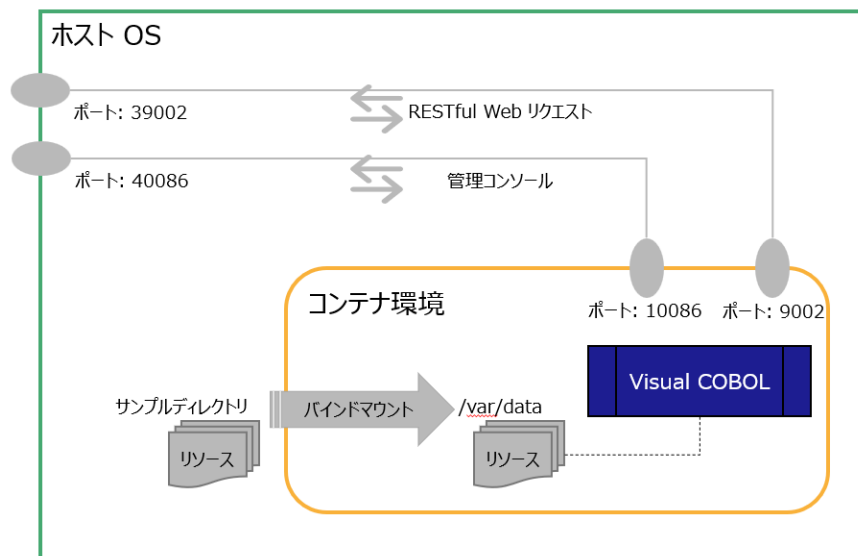
<コンテナコマンド> rmi test

docker rmi test

```
[ec2-user@al2023-v-na ~]$ docker rmi test
Untagged: test:latest
Deleted: sha256:fdfea68572f45e4a4e1df9452951f98f91137d78de122f3a793dd93e42813973
[ec2-user@al2023-v-na ~]$
```

### 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  | リクエスト例                                                                                                                                                                                                                                                                                                                                                                                   |
|--------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 書籍情報検索 | http://localhost:39002/temppath/BookRest/1.0/SearchBook<br>リクエストデータ例)<br><pre>{   "LNK_B_STOCKNO": "1111" }</pre>                                                                                                                                                                                                                                                                        |
| 書籍情報追加 | http://localhost:39002/temppath/BookRest/1.0/AddBook<br>リクエストデータ例<br><pre>{   "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,   } }</pre> |
| 書籍情報削除 | http://localhost:39002/temppath/BookRest/1.0/DeleteBook<br>リクエストデータ例)<br><pre>{   "LNK_B_STOCKNO": "9999" }</pre>                                                                                                                                                                                                                                                                        |

- 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<br>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<br>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<br>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                                     |

- 2) 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:amzn2023\_10.0\_x64

また、2 行目も環境に合わせてコマンドを変更してください。

CONTAINER\_CMD=docker

- 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

```
[ec2-user@al2023-v-na vc-containertutorial02]$ sh setup.sh
6b0fa80fc27f0497ccdf656a51f5f30b1ec0d4d582916c1c2c3c312a165d71e
COBDIR set to /opt/microfocus/VisualCOBOL
Processing -g option...
Copyright 1991-2024 Micro Focus.
Micro Focus Directory Server daemon: Version 1.30.22
Using:
    Repository Type = XML
    Import Path = /var/data/DEMOSV.xml
    Options = 0
    User ID = SYSAD
    Password = [specified]
Response = 0 1.
Import requested processed. Check journal file and export history for full result.
1 server(s) imported.
Processing -s option...
Copyright 1991-2024 Micro Focus.
Micro Focus Directory Server daemon: Version 1.30.22
Request sent...waiting
sha256:7d81473cb5b0f5f81ff4ede74829c574bb71c7057c8614c20742c048aed0050f
mfcs_wk
[ec2-user@al2023-v-na vc-containertutorial02]$
```

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

docker images

```
[ec2-user@al2023-v-na tmp]$ docker images
```

| REPOSITORY          | TAG                     | IMAGE ID     | CREATED        | SIZE   |
|---------------------|-------------------------|--------------|----------------|--------|
| mfcs_bookrest       | latest                  | 7d81473cb5b0 | 16 minutes ago | 1.56GB |
| microfocus/vcdevhub | amzn2023_10.0_x64_login | 5c4a572abc21 | 2 days ago     | 1.61GB |
| microfocus/vcdevhub | amzn2023_10.0_x64       | d424ea524957 | 2 days ago     | 1.56GB |
| amazonlinux         | 2023                    | 4c4d12b6bd64 | 13 days ago    | 144MB  |

- 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

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

mfcs\_bookrest

```
[ec2-user@al2023-v-na vc-containertutorial02]$ docker run --rm -tid --name demo -v $PWD:/var/data:z -p 40086:10086 -p 39002:9002 mfcs_bookrest
1048b2730c3ccbebbc0839915cc7e138e20510ee1cef6b4cd89a4ac0d125d9d0
[ec2-user@al2023-v-na vc-containertutorial02]$
```

- 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"

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

```
[ec2-user@al2023-v-na vc-containertutorial02]$ docker exec -d demo sh -c ". $MFPRODBASE/bin/cobsetenv && escwa -- BasicConfig.MfRequestedEndpoint=tcp:*:10086"
[ec2-user@al2023-v-na vc-containertutorial02]$
```

- 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 && mfdss"

docker exec -d demo sh -c ". \$MFPRODBASE/bin/cobsetenv && mfdss"

- 7) 以下のコマンドで、認証情報を取得します。

<コンテナコマンド> exec demo sh -c '. \$MFPRODBASE/bin/cobsetenv && mfsecretsadmin read microfocus/temp/admin'

docker exec demo sh -c '. \$MFPRODBASE/bin/cobsetenv && mfsecretsadmin read microfocus/temp/admin'

```
[ec2-user@al2023-v-na vc-containertutorial02]$ docker exec demo sh -c '. $MFPRODBASE/bin/cobsetenv && mfsecretsadmin read microfocus/temp/admin'
COBDIR set to /opt/microfocus/VisualCOBOL
{"mfUser":"SYSAD", "mfPassword":"tijiVzlk"}
```

```
[ec2-user@al2023-v-na vc-containertutorial02]$
```

- 8) 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 /u<mfUser> /p<mfPassword>"`

<mfUser>, <mfPassword> の値は前手順で表示された値に置き換えてください。

`docker exec -d demo sh -c ". $MFPRODBASE/bin/cobsetenv && casstart /rDEMOSV /uSYSAD /ptijiVzlk"`

```
[ec2-user@al2023-v-na vc-containertutorial02]$ docker exec -d demo sh -c ". $MFPRODBASE/bin/cobsetenv && casstart /rDEMOSV /uSYSAD /ptijiVzlk"
```

```
[ec2-user@al2023-v-na vc-containertutorial02]$
```

- 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`

```
[ec2-user@al2023-v-na vc-containertutorial02]$ 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"
}[ec2-user@al2023-v-na vc-containertutorial02]$
```

#### 書籍情報追加

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

```
[ec2-user@al2023-v-na vc-containertutorial02]$ curl -X POST http://localhost:39002/temppath/BookRest/1.0/AddBook -d @req/AddBook
{
  "LNK_FILE_STATUS" : "00"
}[ec2-user@al2023-v-na vc-containertutorial02]$
```

#### 書籍情報削除

```
curl -X POST http://localhost:39002/temppath/BookRest/1.0/DeleteBook -d @req/DeleteBook
[ec2-user@al2023-v-na vc-containertutorial02]$ curl -X POST http://localhost:39002/temppath/BookRest/1.0/DeleteBook -d @req/DeleteBookk
{
  "LNK_FILE_STATUS" : "00"
}[ec2-user@al2023-v-na vc-containertutorial02]$
```

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

docker stop demo

```
[ec2-user@al2023-v-na vc-containertutorial02]$ docker stop demo
demo
[ec2-user@al2023-v-na vc-containertutorial02]$
```

- 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

docker rmi mfcs\_bookrest

```
[ec2-user@al2023-v-na vc-containertutorial02]$ docker rmi mfcs_bookrest
Untagged: mfcs_bookrest:latest
Deleted: sha256:7d81473cb5b0f5f81ff4ede74829c574bb71c7057c8614c20742c048aed0050f
Deleted: sha256:46144923108e0113b79a1e01c6da2c342d14cfb0b37b0eea772e62392beabeb
[ec2-user@al2023-v-na vc-containertutorial02]$
```

## 4 補足

### 4.1 サンプルスクリプトについて

#### 4.1.1 setup.sh

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。

```

1 VC_PROD_IMAGE=microfocus/vcdevhub:amzn2023_10.0_x64
2 CONTAINER_CMD=docker
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 && mfd"
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

```

| 行番号   | 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  . $MFPRODBASE/bin/cobsetenv
3  credential=`mfsecretsadmin read microfocus/temp/admin`
4  user=$(echo $credential | grep -oP "mfUser":"¥K[^"]+')
5  password=$(echo $credential |grep -oP "mfPassword":"¥K[^"]+')
6  mfd /g 5 /var/data/DEMOSV.xml O $user $password
7  cd /var/data/deploy && mfdepinst BookRest.car
8  mfd /s 1 $user $password
9  while :
10 do
11   PROCINF=`ps -efa |grep mfd |grep -v grep`
12   if [ "$PROCINF" == "" ]; then
13     break
14   fi
15   sleep 1
16 done
17 exit 0

```

| 行番号  | 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。                                                                         |
| 6～8  | 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。 |
| 9～16 | 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。                                                                                      |

#### 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。