HP Servicegurad による
Micro Focus Enterprise Server
クラスタ化の検証報告

日本ヒューレット・パッカード株式会社
MFA ビジネス推進部
マイクロフォーカス株式会社
技術部

目次

はじめに

本書では、Micro Focus Enterprise Server（以下、Enterprise Server)で提供するアプリケーションサービスをHP Serviceguard（以下、Serviceguard)により共有ディスクを利用したクラスタ上で動作させた検証を報告します。

Enterprise Server は Windows、Unix、Linux 環境で COBOL サービスをトランザクショナルに管理・実行するアプリケーション サーバです。Enterprise Server は Micro Focus Net Express および Server Express COBOL コンパイラ ツール群の運用部分です。Enterprise Server はメインフレームのトランザクション環境と同等の OLTP 機能を、J2EE アプリケーションサーバと協調実行できるよう設計されています。ユーザーにとっては、ハイエンドのトランザクション システムで、機能およびパフォーマンスと同様に重要なことは、その環境がきわめて可用性、信頼性、保守性に優れていることです。

Serviceguard は、豊富な実績をもつハイアベイラビリティ･ミドルウェアです。さまざまなハードウェア/ソフトウェアの障害から基幹系アプリケーションを保護する高度な機能を HP-UX 上で提供します。Serviceguard は、最大 16 個のノードを疎結合することでエンタープライズ･クラスタを構成し、LAN で接続されたクライアントに対して可用性の高いアプリケーション･サービスを提供できます。

ServiceguardとEnterprise Serverを組み合わせ、如何に信頼性・可用性の高いCOBOLサービスシステムを提供していくか、その方法について検証します。

注意事項

本書に記載されている内容は、検証環境での評価システムでの情報であり、実際のシステム環境では、それぞれ個別に評価をし、固有の設定が必要になります。本書に記載されている内容は、参考値であり、実際のシステムでの動作を保証するものではありません。

使用環境

使用したハードウェア

(サーバ)
HP9000 rp3440　　２台
PA8900 1GHz/64MB dualcore CPU x 2 搭載　(OSとしては4CPUとして認識)
8GB DDR メモリ (4x2GB) x 3 搭載
300GB-10K U320 SCSI Disk x 3 搭載

(ストレージ)
HP EVA8000-A 2C1D Array　　１台
146GB 10Krpm 2G FC-AL HDD x 32搭載

使用したソフトウェア

(OS)
HP-UX 11i v2 (11.23)

(アプリケーションサーバ)
Micro Focus Enterprise Server 4.0J Service Pack 2 for PA-RISC HP-UX

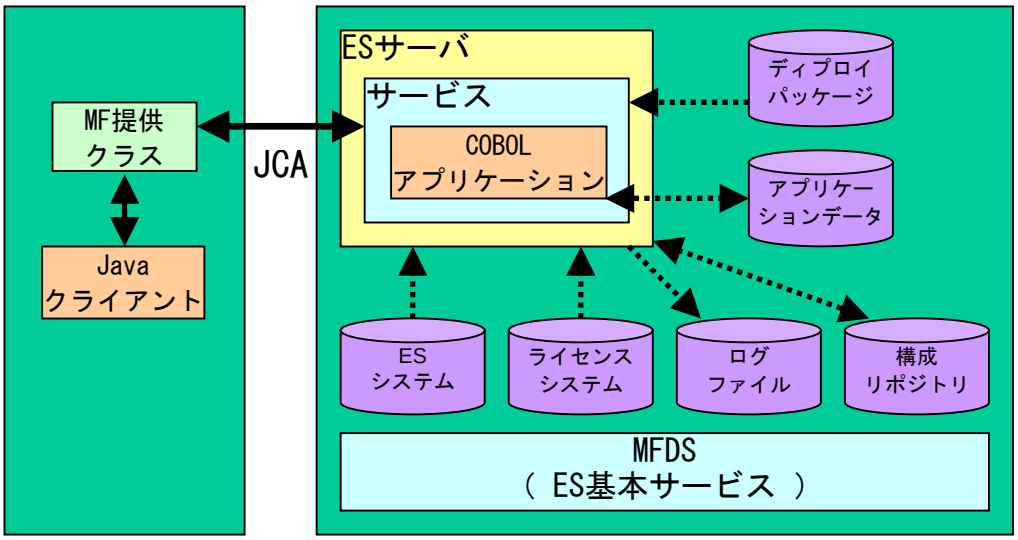
(クラスタリング)
HP Serviceguard A.11.17.00

(言語)
Micro Focus Server Express 4.0J Service Pack 2 for PA-RISC HP-UX
Java(TM) 2 SDK, Standard Edition 1.4.2_8

概要

Enterprise Server とテストシステム

今回のテストで Enterprise Server に実装するサンプルモデルを用意しました。



Java クライアントから要求されたリクエストは Enterprise Server 上にディプロイされたサービスの COBOL アプリケーションに引き渡されます。この COBOL アプリケーションは索引ファイルであるアプリケーションデータから連番を取得し、この連番をカウントアップして書き込みします。そして日時、プログラム名、実行ホスト名、連番を返します。Java クライアントはこれを表示して終了します。

以下に関連する Enterprise Server の構成要素について説明します。

（プロセス関連）

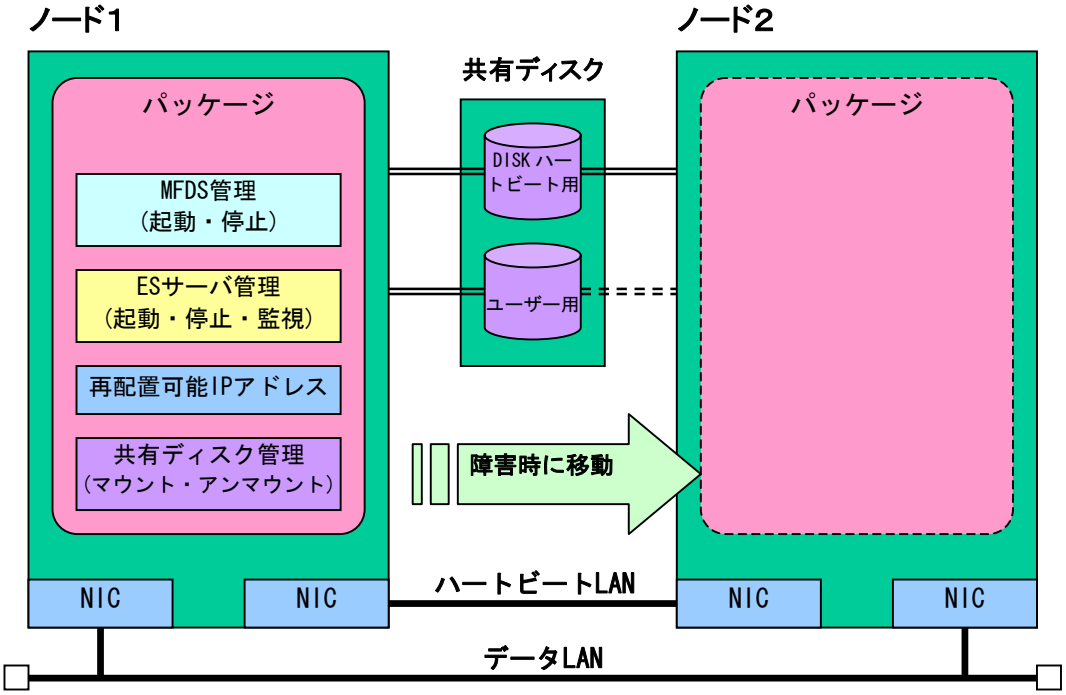
- MFDS　Enterprise Server の基本サービスの一つ。1 ノードに 1 つだけ起動が必要
- ES サーバ　Enterprise Server の管理単位。1 つのノードに複数定義でき、複数のサービスを含むことができる。
- サービス　Enterprise Server で提供する COBOL サービスであり、ディプロイの単位。
- COBOL アプリケーション　COBOL プログラムが提供する業務プロセス。
- MF 提供クラス　クライアントから Enterprise Server に接続するために利用
- Java クライアント　コマンドライン起動で Enterprise Server にリクエストを出し受け取った結果を表示する。

（ファイル関連）

- ES システム　Enterprise Server のシステムコマンド、ライブラリなどのファイルを含む。$COBDIR 環境変数がポイントするディレクトリパス。共有可。
- ライセンスシステム　Enterprise Server のライセンス管理の製品のライセンスが投入されるデータベースを保管する。このディレクトリをディスクコピーなどで移動・復元することはでない。共有不可。

・ログファイル　　Enterprise Server のログ。ES サーバ単位で管理され、再起動によって自動的に新しいファイルに書き換えられる。共有可。
・構成リポジトリ　Enterprise Server の設定情報を格納したファイル群。共有可。
・ディプロイパッケージ
car ファイル。ディプロイに必要なファイル群。実装する際のディプロイパッケージは Interfase Mapping Tool Kit(IMTK)により COBOL ソースから容易に生成することが出来る。Server Express によってコンパイルされた実行形式のファイルを含む。（ソースサンプルは巻末に記載）共有可。
・アプリケーションデータ
COBOL プログラムが利用するユーザファイル。共有可。
・Java クライアント Java プログラム。（ソースサンプルは巻末に記載）
・MF 提供クラス　クライアントプログラムが利用する Enterprise Server 接続用の Java クラス。

クラスタリング



Serviceguard ではフェイルオーバーする一連の操作をパッケージの単位で管理します。

パッケージには、再配置可能 IP アドレス(仮想 IP アドレス)や共有ディスクパーティションの有効化、アプリケーションの起動・停止・監視の処理を設定します。
再配置可能 IP アドレスや共有ディスクパーティションはパッケージが立ち上がると自動的に有効化され停止すると無効化します。
MFDS や ES サーバなどの起動・停止はそれぞれのシェルにコマンドで処理を記述します。
本運用ではこの起動・停止のシェルには障害時の状態回復・状態保存を行なう処理なども組み込む必要があります。
またアプリケーションの監視は、障害とする事象が発生するまで監視を繰り返す業務監視用のシェルをユーザーが準備しこれを組み込む事で行ないます。

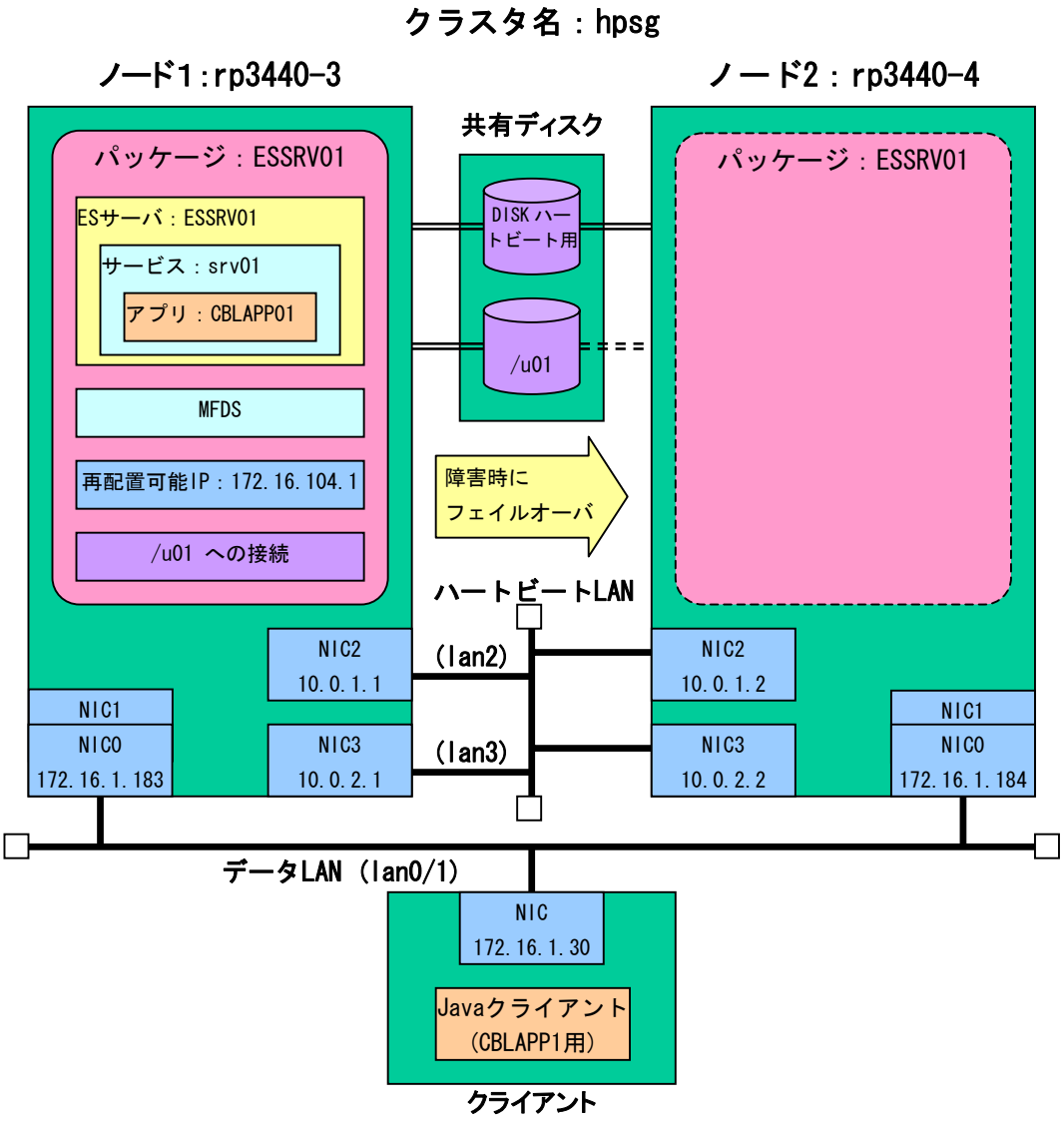
環境設定

下図のモデルを想定した場合の環境の Serviceguard、MF 製品の設定例を記述します。

手順

１．事前準備
２．製品インストールと動作環境の設定
３．Serviceguard の設定
４．COBOL コンポーネントの準備
５．Enterprise Server の設定
６．クライアントの作成

環境設定モデル



１．事前準備

各ノードに搭載された NIC の設定、共有ファイルの接続に関する事前の設定、その他の OS レベルの設定を済ませておきます。

２．製品のインストール

（クラスタ環境）
各ノード個別に下記２つの製品をインストールします。
・Micro Focus Enterprise Server 4.0J Service Pack 2 for PA-RISC HP-UX
・HP Serviceguard A.11.17.00

（開発環境）
実装する COBOL コンポーネントの準備に COBOL・Java 開発環境が必要になります。このノード上もしくは別の同一 OS のマシン上に下記の製品をインストールします。
・Micro Focus Server Express 4.0J Service Pack 2 for PA-RISC HP-UX
・Java(TM) 2 SDK, Standard Edition 1.4.2_8
・j2ee.jar

（クライアント環境）
本検証ではクライアントは Java プログラムからリクエストしますので、クライアントに Java 環境が必要になります。本検証では
・Java(TM) 2 SDK, Standard Edition 1.4.2_8
・j2ee.jar

またあらかじめ各製品の動作に必要な環境変数等の設定をしておきます。

３．Serviceguard の設定

本検証では Serviceguard の設定は各種設定ファイルを作成し、コマンドにより反映しました。ここで設定するのはクラスタ構成ファイル、パッケージ構成ファイル、パッケージ制御スクリプトです。

クラスタ構成ファイル
クラスタ各ノードとネットワーク構成を登録します。今回は２ノードの設定をしました。
（/etc/cmcluster/cmclconf.ascii）

パッケージ構成ファイル
パッケージ定義を必要数準備します。
（/etc/cmcluster/mfpkg1/ESSRV01.conf）
（/etc/cmcluster/mfpkg2/ESSRV02.conf）

パッケージ制御スクリプト
このパッケージに再配置可能 IP アドレス、共有ディスク、フェイルオーバーするサービス、監視プロセスなどを登録します。
（/etc/cmcluster/mfpkg1/ESSRV01.cntl）
（/etc/cmcluster/mfpkg2/ESSRV02.cntl）

本検証ではフェイルオーバーサービスの起動停止、監視プロセスの立ち上げは後述の mf.sh で記述する事にします。

cntl ファイルは OS コマンドでコピー、conf ファイルは Serviceguard のコマンド(cmcheckconf、cmapplyconf)にてチェック/配布しました。

次に ES サーバを管理するコマンドを設定します。以下は代表的なコマンド群です。mfds(Micro Focus Directory Server)は Enterprise Server の基本サービスであり、１ノードに１つのみ起動できる必須サービスです。mfds は ES サーバの立ち上げを行なう前に起動しておく必要があります。

処理	コマンド例
mfds の初期化処理 (必要に応じて)	ps -ef | grep mfds > $$tmp.txt 2> /dev/null tmp=`cat $$tmp.txt | grep -v "grep -E "| awk '{print $2}'` if [".$tmp" != "."] ; then 　　kill -9 $tmp fi rm $$tmp.txt
mfds の起動	mfds & Sleep 5
mfds の停止	ps -ef | grep mfds > $$tmp.txt 2> /dev/null tmp=`cat $$tmp.txt | grep -v "grep -E "| awk '{print $2}'` if [".$tmp" != "."] ; then 　　kill -9 $tmp fi rm $$tmp.txt
ES サーバの初期化処理 (必要に応じて)	para=<ES サーバ名> ps -ef | grep $para | grep -E "cassi|cascd|casjcp|casmgr|mfcs " > $$tmp.txt 2> /dev/null tmp=`cat $$tmp.txt | grep -v "grep -E "| awk '{print $2}'` if [".$tmp" != "."] ; then 　　kill -9 $tmp fi rm $$tmp.txt
ES サーバの起動	casstart /r<ES サーバ名>
ES サーバの停止	casstop /r<ES サーバ名>

パッケージで管理する ES サーバの関連処理をシェル mf.sh として記述します。

(Enterprise Server 管理シェル例 1：mf.sh)

```
#!/usr/bin/sh
mffunc=$1
host_name=$2
es_name=$3
APPFILE01=/u01/APPFILE01
export APPFILE01
APPFILE02=/u02/APPFILE02
COBMODE=32
export COBMODE
export APPFILE02
COBDIR=/opt/mf/ES40SP2
export COBDIR
SHLIB_PATH=$COBDIR/lib
export SHLIB_PATH
PATH=$COBDIR/bin:$PATH
export PATH

case $mffunc in
start)
/etc/cmcluster/mfpkg1/killmfds.sh >> /etc/cmcluster/mfpkg1/ESSRV01.cntl.log
/etc/cmcluster/mfpkg1/killallcas.sh ESSRV01 >> /etc/cmcluster/mfpkg1/ESSRV01.cntl.log
mfds &
/wait 5
casstart /rESSRV01
exit
;;
monitor)
sleep 10
mdump_func()
{
mdump $host_name $es_name > $$tmp.txt  2> /dev/null
web_srv_flg=0
cat $$tmp.txt | while read line
do
          # if mfds not started "SERVER_DOWN" found
          # actual text is 'm_ldap_bind: 0x51 SERVER_DOWN'
        tmp=`echo $line | fgrep "SERVER_DOWN"`
        if [ ".$tmp" != "." ] ; then
                    # echo $line
                return 3
        fi

        tmp=`echo $line | fgrep "CN:"`
        if [ ".$tmp" != "." ] ; then
                tmp=`echo $line | fgrep "Web Services"`
                if [ ".$tmp" != "." ] ; then
                        # echo $line
                        web_srv_flg=1
                else
                        # echo "no " $line
                        web_srv_flg=0
                fi
        fi

        if [ $web_srv_flg -eq 1 ] ; then
                lisnr=`echo $line | fgrep "mfListenerStatus:"`
                if [ ".$lisnr" != "." ] ; then
                        lisnr=`echo $line | fgrep "Started"`
                        if [ ".$lisnr" != "." ] ; then
                                return 0
                        else
                                # echo $line
                                return 1
                        fi
                fi
        fi
done
}
```

```
if [ ".$host_name" = "."  ] || [ ".$es_name" = "." ] ; then
          host_name="localhost"
          es_name=""
fi
while : ; do
        # echo "monitoring"
        mdump_func        res=$?
        rm $$tmp.txt
        if [ "$res" = "1" ]; then
                echo "**** MF Service Failed  ****" >> /etc/cmcluster/mfpkg1/ESSRV01.cntl.log
                exit 0
        fi
        if [ "$res" = "3" ]; then
                echo "**** MF mfds Down  ****" >> /etc/cmcluster/mfpkg1/ESSRV01.cntl.log
                exit 0
        fi
        sleep 5
done
;;
halt)
casstop /rESSRV01
sleep 5
/etc/cmcluster/mfpkg1/killmfds.sh >> /etc/cmcluster/mfpkg1/ESSRV01.cntl.log
exit
esac
```

start) や **halt)** で記述の部分はパッケージの起動時・停止時に必要なサービスの管理を行ないます。本検証ではテストパターンによって処理の書き換えを行いました。

Monitor) で記述の部分はマイクロフォーカスが提供する mdump ユーティリティにて、Enterprise Server のステータスを取得し、これを解析してサービスが正常に提供されていることを監視する処理です。対象のサービスが起動状態でなくなるまで監視し続けます。Serviceguard はこの監視処理が終了したことでアプリケーション障害と見なし、フェイルオーバーを開始することになります。

また、万が一システムが中途半端に停止した状態から再起動する場合に備え、各サービス起動の前に不要プロセスを kill する処理を組み込みます。

(Enterprise Server 管理シェル例 2：killmfds.sh)

```
ps -ef | grep mfds  > $$tmp.txt  2> /dev/null
cat $$tmp.txt
tmp=`cat $$tmp.txt | grep -v "grep -E "| awk '{print $2}'`
if [ ".$tmp" != "." ] ; then
        echo "kill -9 " $tmp
        kill -9 $tmp
fi
rm $$tmp.txt
```

(Enterprise Server 管理シェル例 3：killallcas.sh)

```
para=$1
ps -ef | grep $para | grep -E "cassi|cascd|casjcp|casmgr|mfcs" > $$tmp.txt  2> /dev/null
cat $$tmp.txt
tmp=`cat $$tmp.txt | grep -v "grep -E "| awk '{print $2}'`
if [ ".$tmp" != "." ] ; then
        echo "kill -9 " $tmp
        kill -9 $tmp
fi
rm $$tmp.txt
```

４．COBOL コンポーネントの準備

Enterprise Server に実装するコンポーネントの準備を行ないます。
作業は開発環境にて行ないます。本検証では Server Express が提供する Interface Mapping Tool Kit を使って一連の作業を簡略化しています。
初めに COBOL ソースの元にサービスに対するデフォルトのマッピングを作成します。サービス名は任意に命名します。

```
imtkmake -defmap src=cblapp01.cbl service=srv01 type=ejb
```

これで”srv01.xml”の名称でデフォルトのマッピング情報が生成されます。
デフォルトではこのサービスのパラメータはプログラム中のパラメータ名で全て”io”(入出力)として生成されています。本サンプルでは出力パラメータのみであるため実態に合わせ全て”out”に手修正しました。
そして次のコマンドでディプロイパッケージを生成します。

```
imtkmake -generate service=srv01 type=ejb
```

これで”srv01.car”ファイルが生成されます。

更にここで同じ COBOL ソースを cob コマンドでコンパイルし、実行形式ファイル(本検証では gnt 形式を選択)を生成しておきます。

５．Enterprise Server の設定

まず/etc/services に次の 2 行を追加します。

```
mfcobol         86/tcp
mfcobol         86/udp
```

次に mfds コマンドで Enterprise Server を起動します。
Web ブラウザから http://(IP アドレス):86 でアクセスすると Enterprise Server Administration 画面が開きます。ここで多くの操作を実行できる様になります。

初めにデフォルトで生成されている ES サーバ”ESDEMO”を削除します。
次に新規に必要な ES サーバを作成します。名称以外はデフォルトの設定としました。

次にディプロイパッケージを使ってサービスの登録を行ないます。
<Enterprise Server インストールディレクトリ>/deploy の下に任意のディプロイ用ディレクトリを作成し、その下に４で作成した car ファイルと実行形式ファイルをコピーします。

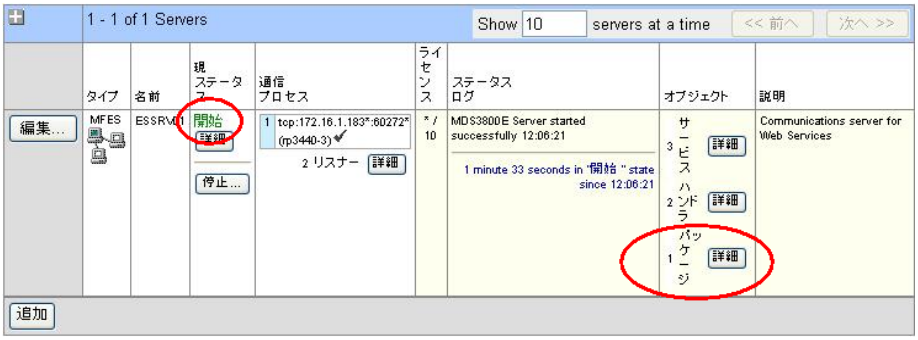
<Enterprise Server インストールディレクトリ>/deploy/.mfdeploy を以下の様に書き換えます。

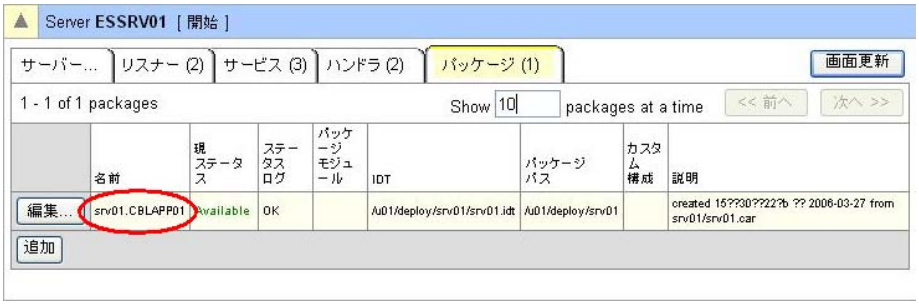
```
MFDS=localhost
MFUS_SERVER=ESSRV01
MFUS_LISTENER=Web Services
```

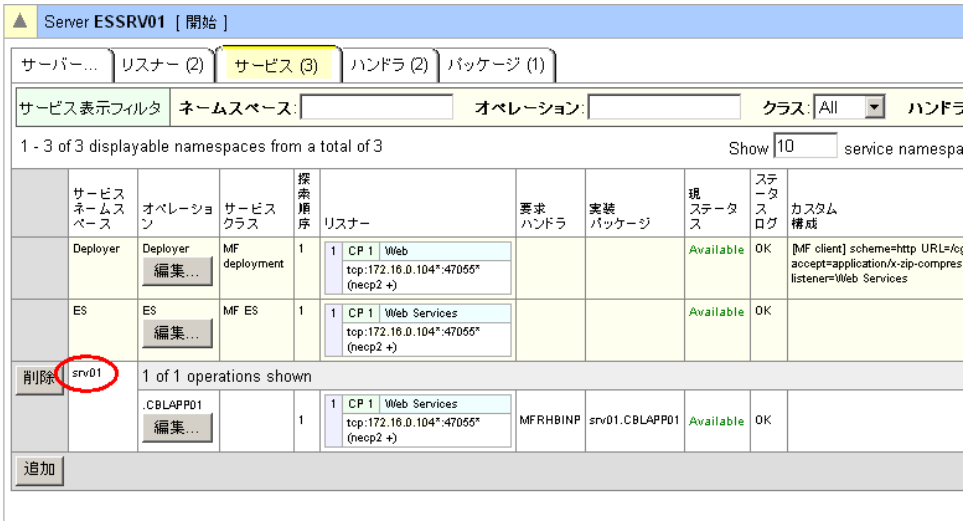
<Enterprise Server インストールディレクトリ>/deploy の下のディプロイ用ディレクトリに移動して下記のコマンドを実行します。

```
mfdepinst srv01.car
```

すると.mfdeploy に記載した ES サーバ上に car ファイル作成時に指定した名称でサービスが作成されます。登録された ES サーバは開始(Start)ボタンで手動で開始できます。







次にリスナーの設定をします。
ES サーバを停止後、リスナーの画面の編集(Edit)ボタンで編集画面を開き、”WebServices”の中に”*:*”で設定されている IP アドレス:ポート番号を固定で設定します。

ここで設定する IP アドレスはパッケージとしてフェイルオーバーする場合には再配置可能 IP(172.16.104.1)を、パッケージ化せずに単体でテストする場合には定常 IP(172.16.1.183)を設定すると便利です。

６．クライアントの作成

本検証で Enterprise Server 上にディプロイしたサービスにリクエストを出すのは Java のクライアントプログラムです。（ソースサンプルは巻末に記載）

このプログラムを利用するには$CLASSPATH に下記の 3 ファイルを指定する必要があります。

j2ee.jar
mfconnector.jar　　（<Enterprise Server インストールディレクトリ>/lib に存在）
mfcobolpure.jar　　（<Enterprise Server インストールディレクトリ>/lib に存在）

検証

テストパターン

本検証では以下の 10 パターンを確認しました。

パターン	タイトル	説明
①	手動切り替え	Serviceguard への操作による切り替えを確認
②	データ LAN 両系(二重)障害	稼動側サーバに接続されたデータ LAN のケーブルを 2 本とも引き抜いて切り替えを確認
③	業務障害	業務監視用シェルの異常検出による切り替えを確認
④	OS のみで待機 (シングルスタンバイ)	待機状態のノードでは MFDS 起動せず切り替えを確認
⑤	MFDS 起動で待機 (高速スタンバイ 1)	待機状態のノードでは MFDS 起動して切り替えを確認
⑥	DS サーバ起動で待機 (高速スタンバイ 2)	待機状態のノードでは MFDS、DS サーバを起動状態で切り替えを確認
⑦	別の業務を提供しながら待機 (相互スタンバイ)	待機状態のノードでは MFDS 起動、切り替える DS サーバとは別の DS サーバを動作させせながら切り替えを確認。ノード 1⇔ノード 2 の双方向で確認
⑧	ディプロイパッケージを共有化	ディプロイするパッケージを共有化し、切り替え後に同じパッケージを引き継ぐことを確認
⑨	リポジトリを共有化	MFDS のリポジトリファイルを共有化し、切り替え後に Enterprise Server の管理情報を引き継ぐことを確認
⑩	ログ出力先を共有化	ログファイルの出力先を共有化し、切り替え後のログ照会ができる事を確認

①～③はフェイルオーバーのトリガーのテストです。代表的なトリガー3 種類のパターンで確認します。
フェイルオーバーをステータスで確認します。

④～⑦は待機側のノードの状態を変えるテストです。本検証では Enterprise Server のプロセスの実行状態を変えて確認します。
Java クライアントから繰り返しリクエストを出し続け、フェイルオーバー後もサービスが提供され続ける事を確認します。

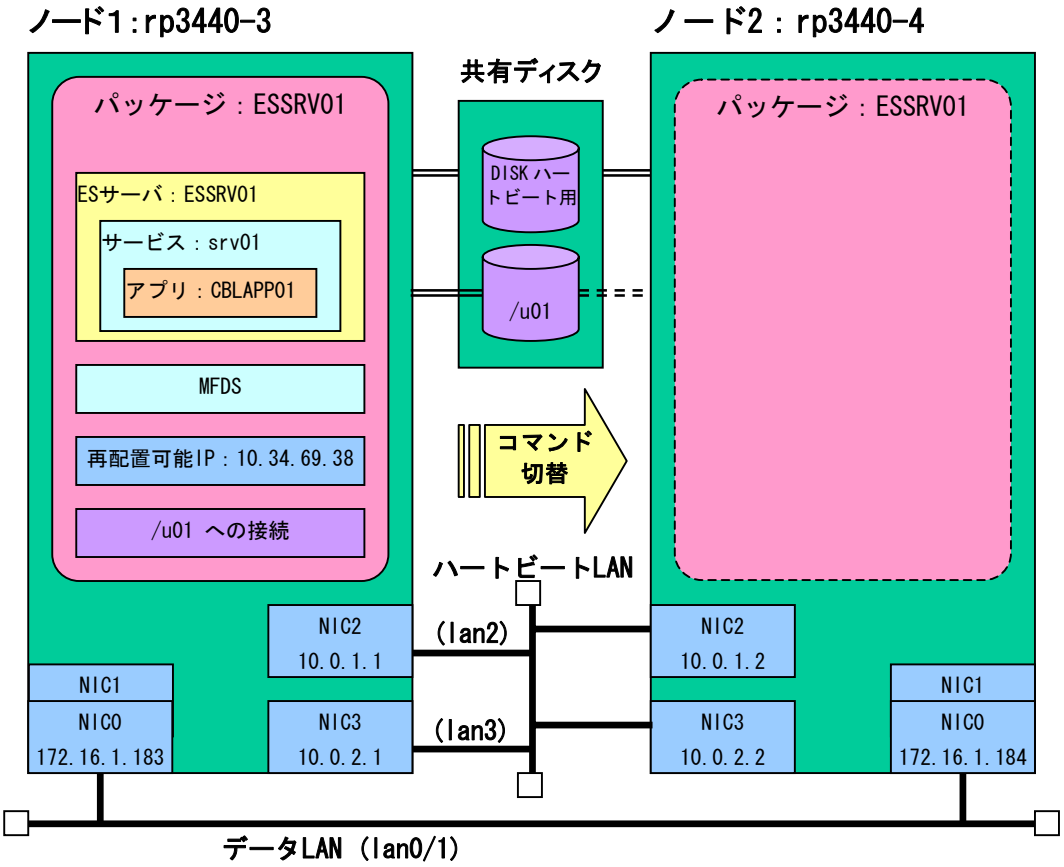
⑧～⑩は Enterprise Server システムが利用する情報を共有化するテストです。

以降、パターン毎の設定と検証結果を記述します。

パターン① 手動切り替え

＜目的＞
通常の運用でも行なわれる動作の確認。
Serviceguard のコマンドを実行し、パッケージの移動が行なわれるか検証する。

＜構成＞



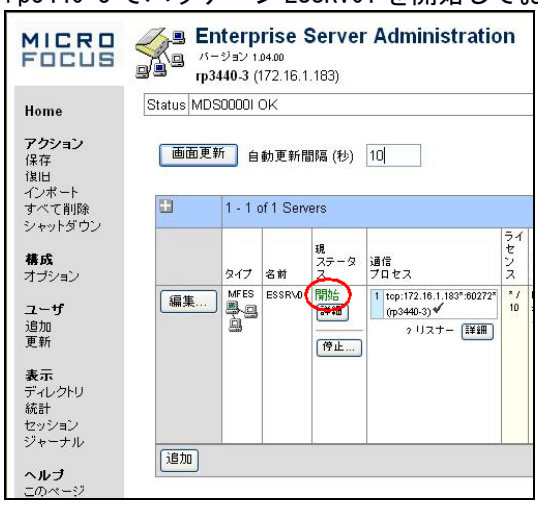
＜設定のポイント＞
ES 管理シェル mf.sh の start) に組み込む起動コマンド

```
killmfds.sh
killallcas.sh ESSRV01
mfds &
/wait 5
casstart /rESSRV01
```

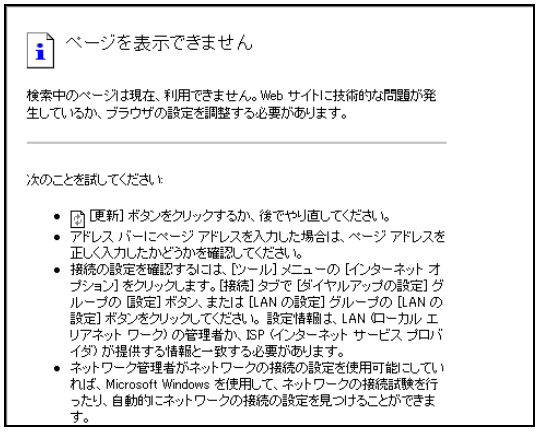
ES 管理シェル mf.sh の halt)に組み込む停止コマンド

```
casstop /rESSRV01
sleep 5
killmfds.sh
```

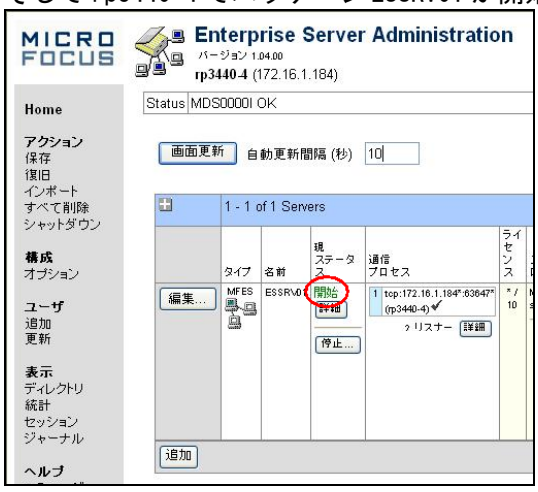
＜検証＞
このパターンでは Serviceguard コマンドにてフェイルオーバーを行なう。
rp3440-3 でパッケージ ESSRV01 を開始しておく。



次に Serviceguard コマンドにてパッケージ停止する。
　＞ cmhaltpkg ESSRV01
このコマンドによりパッケージの移動処理が開始する。
すると、パッケージ ESSRV01 の停止処理により MFDS が止まるため rp3440-3 の管理画面が表示できなくなる。



そして rp3440-4 でパッケージ ESSRV01 が開始される。

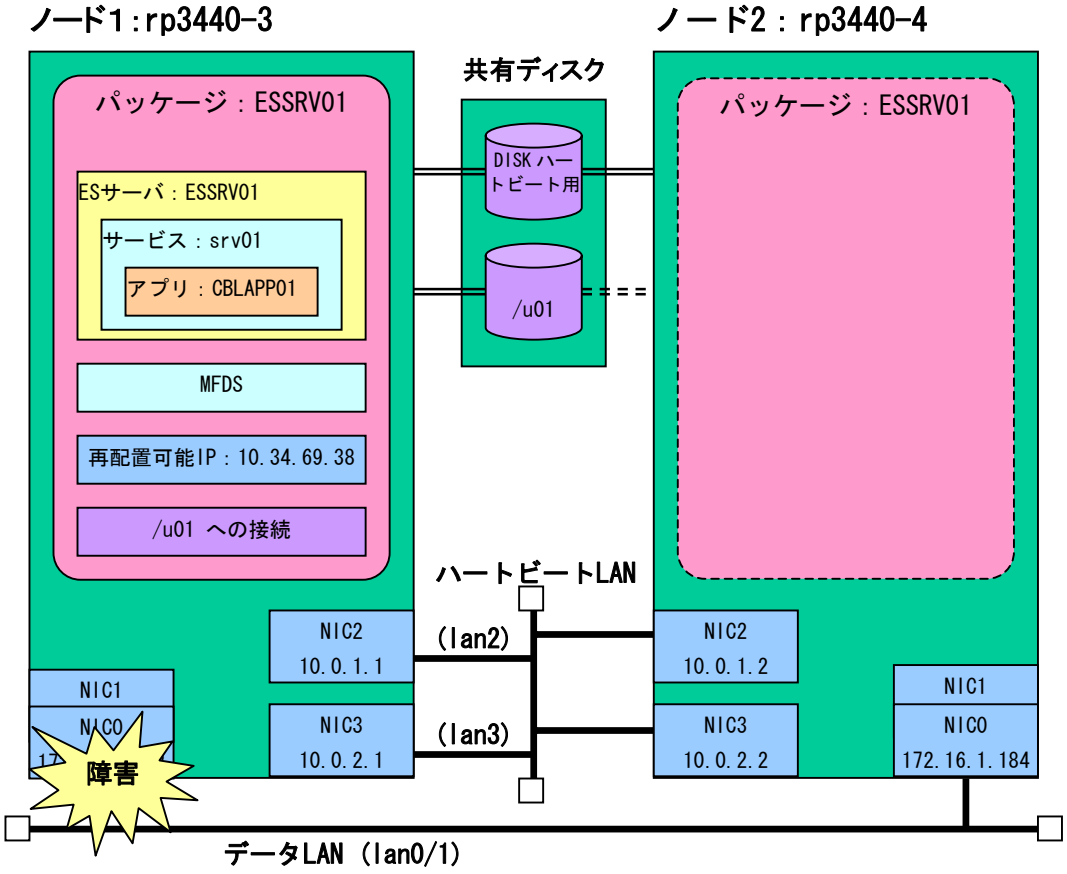


＜結果＞
Serviceguard のコマンド cmhaltpkg によるパッケージの移動を確認。

パターン② データ LAN 両系(二重)障害

＜目的＞
Serviceguard の機能のみで検知する障害パターンの確認
稼動側サーバに接続されたデータ LAN の両系(二重)障害を起こし、パッケージの移動が行なわれるか検証する。

＜構成＞



＜設定のポイント＞
ES 管理シェル mf.sh の start) に組み込む起動コマンド

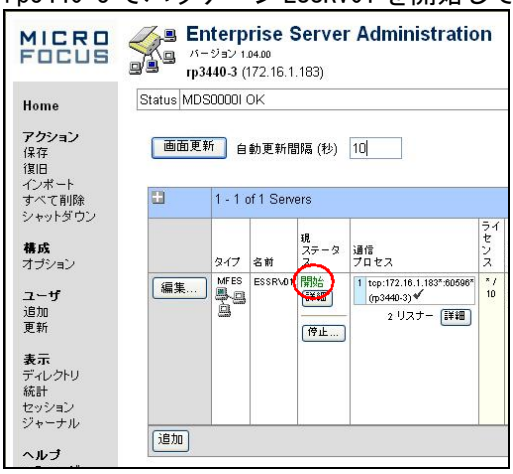
```
killmfds.sh
killallcas.sh ESSRV01
mfds &
/wait 5
casstart /rESSRV01
```

ES 管理シェル mf.sh の halt) に組み込む停止コマンド

```
casstop /rESSRV01
sleep 5
killmfds.sh
```

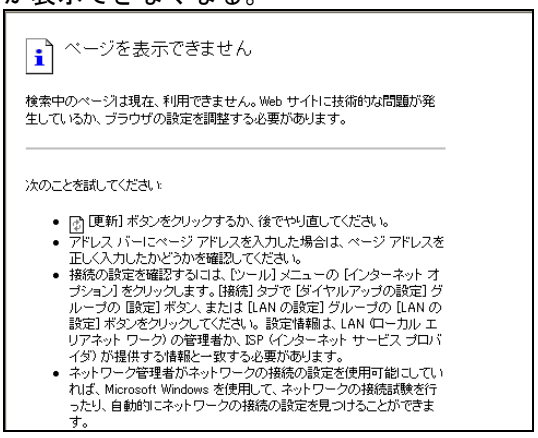
クラスタ構成ファイル cmclconf.ascii にてデータ LAN 両系(二重)の定義をしておく

＜検証＞
rp3440-3 でパッケージ ESSRV01 を開始しておく。

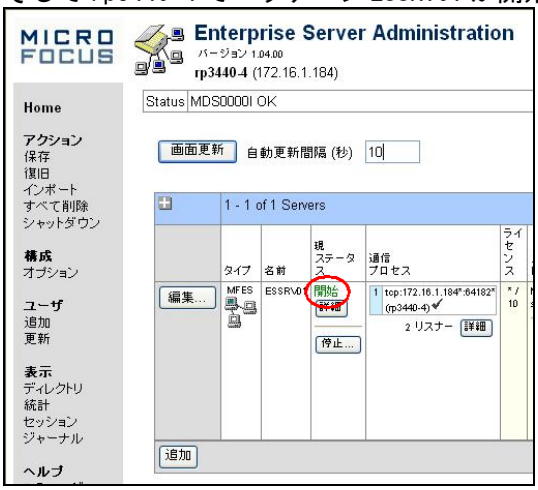


稼動側サーバに接続されたデータ LAN のケーブルを 2 本とも引き抜く。

すると、パッケージ ESSRV01 の停止処理により mfds が止まるため rp3440-3 の管理画面が表示できなくなる。



そして rp3440-4 でパッケージ ESSRV01 が開始される。

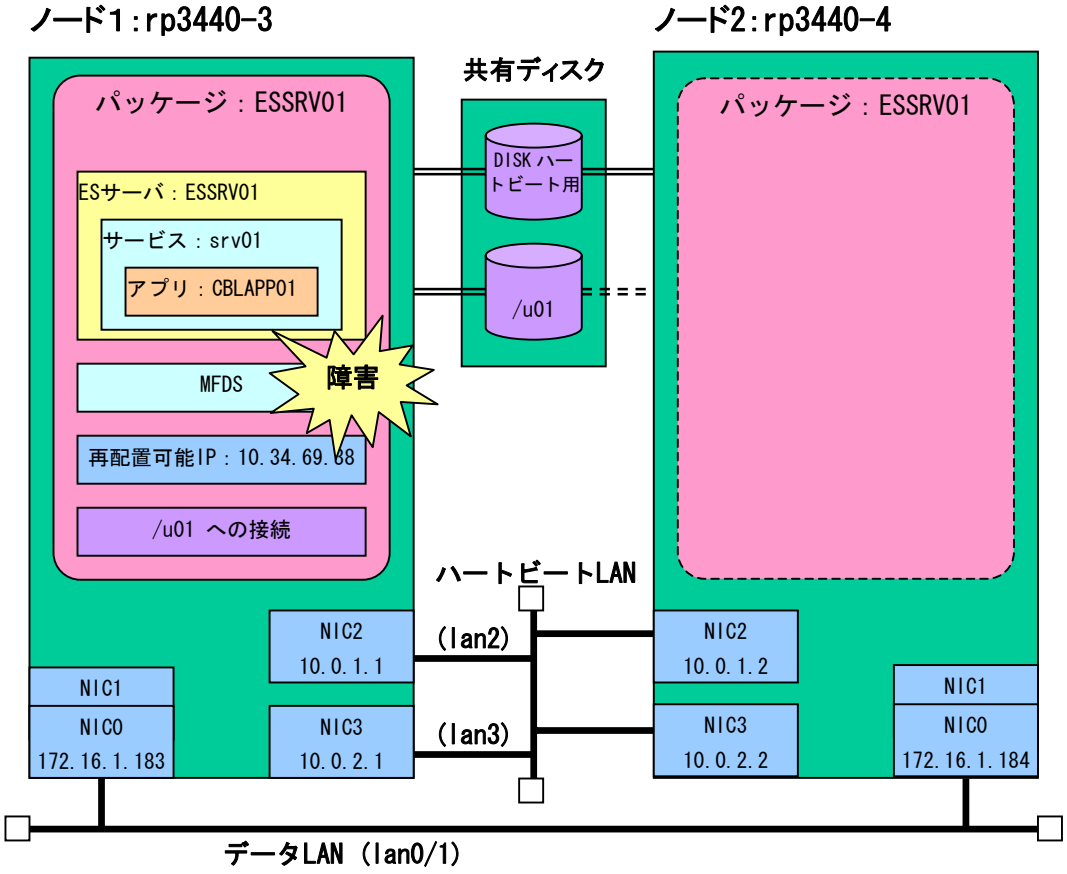


＜結果＞
データ LAN 両系(二重)障害によるパッケージの移動を確認。

パターン③ 業務障害

＜目的＞
ユーザー作成の監視シェルにより検知する障害パターンの確認
本検証では Enterprise Server のプロセス異常を想定し、主要サービスである MFDS を停止することでパッケージの移動が行なわれるか検証する。

＜構成＞



＜設定のポイント＞
ES 管理シェル mf.sh の start) に組み込む起動コマンド

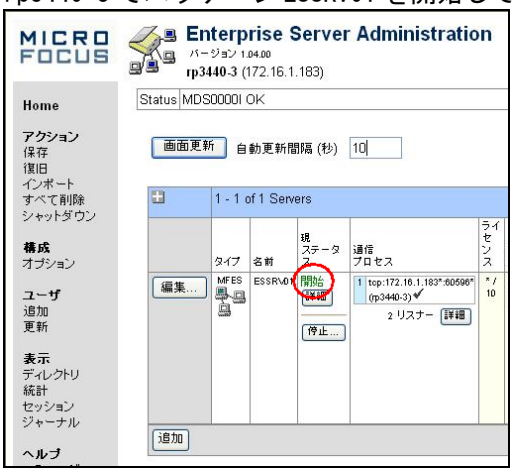
```
killmfds.sh
killallcas.sh ESSRV01
mfds &
/wait 5
casstart /rESSRV01
```

ES 管理シェル mf.sh の halt) に組み込む停止コマンド

```
casstop /rESSRV01
sleep 5
killmfds.sh
```

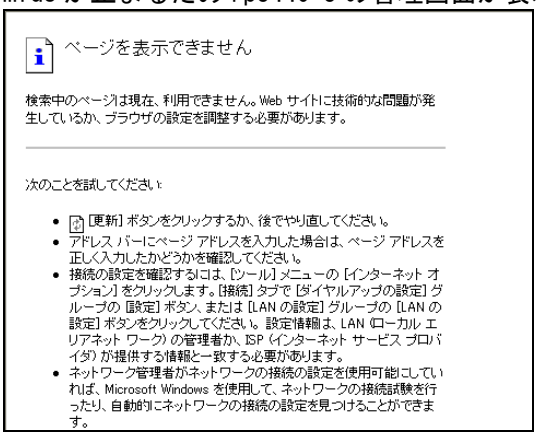
cntl ファイルにて監視用のシェルを起動するように設定
本検証では ES 管理シェル mf.sh の monitor) で記述した監視処理を起動した。
このシェルは自動的に監視対象をチェックし続け、異常を検知した時に終了するように作成する。この監視処理は mfdump の出力の Web Service のステータスにて異常を検知する方式とした。

＜検証＞
rp3440-3 でパッケージ ESSRV01 を開始しておく。

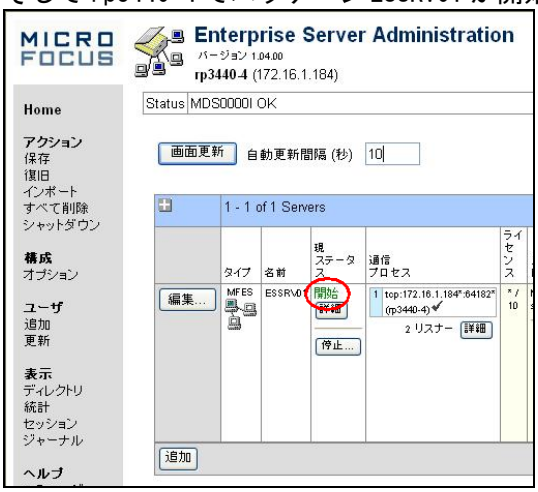


次に MFDS のプロセスを kill する。
> kill -9 <mfds32 の pid>

すると、ES 管理シェル mf.sh の **monitor)**が障害を検知しフェイルオーバーが開始する。
mfds が止まるため rp3440-3 の管理画面が表示できなくなる。



そして rp3440-4 でパッケージ ESSRV01 が開始される。

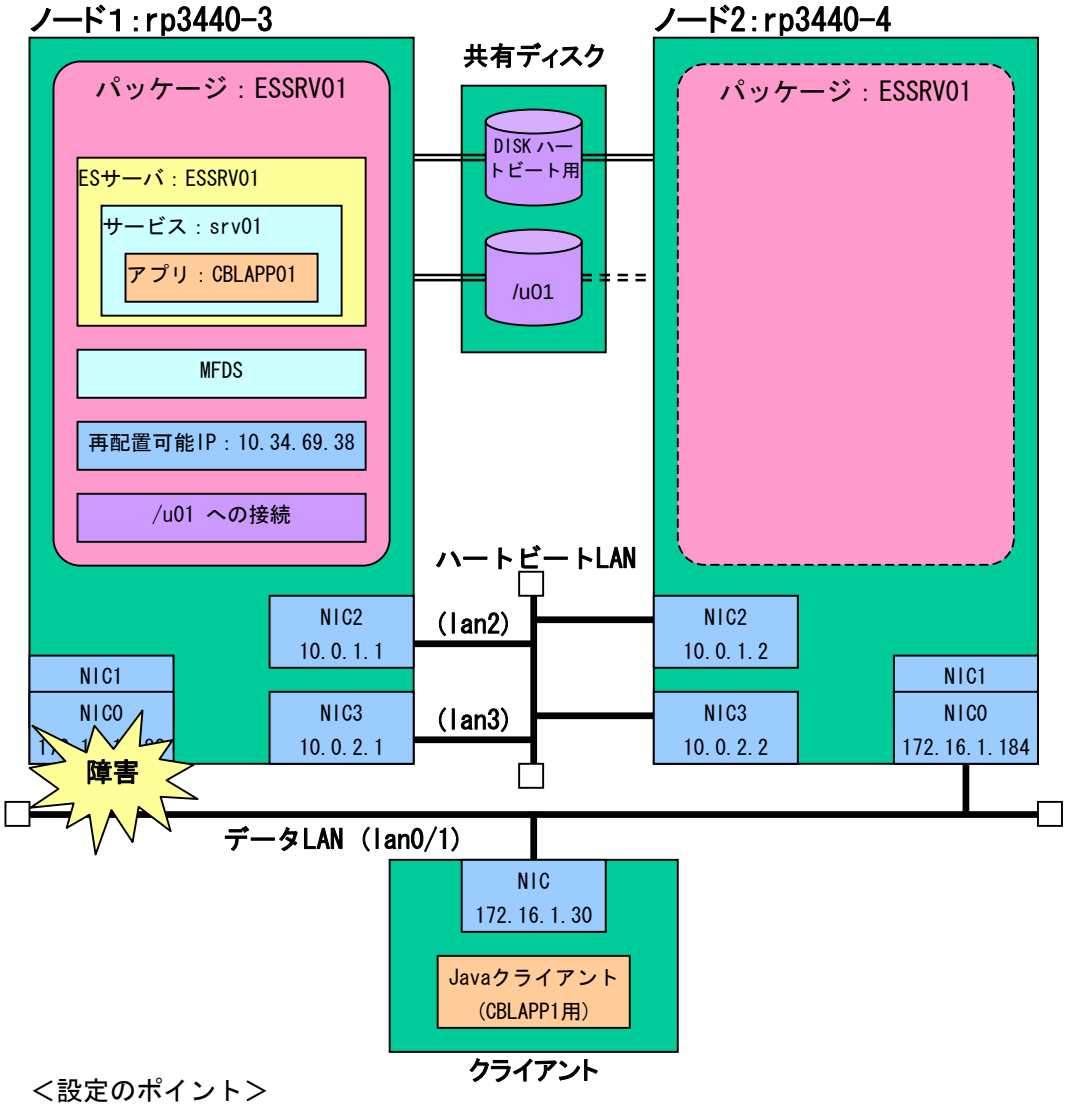


＜結果＞
ユーザー作成の監視シェルにより検知した障害によるパッケージの移動を確認。

パターン④ OS のみで待機

＜目的＞
待機側のノードで Enterprise Server の関連プロセスは起動しない状態からのフェイルオーバーを確認する。
フェイルオーバー前と後で同じサービスが継続される事も確認する。

＜構成＞



＜設定のポイント＞
（パターン①～③の構成と同じ）
ES 管理シェル mf.sh の **start)** に組み込む起動コマンド

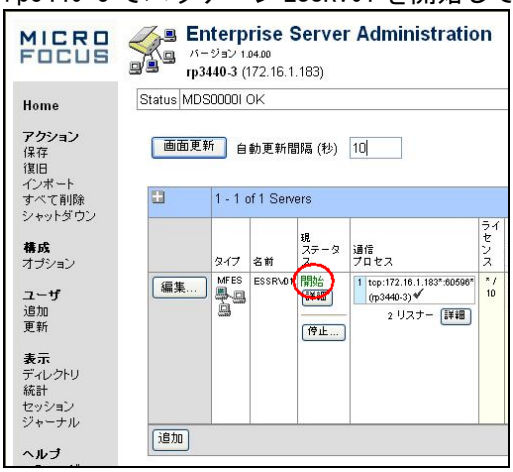
```
killmfds.sh
killallcas.sh ESSRV01
mfds &
/wait 5
casstart /rESSRV01
```

ES 管理シェル mf.sh の **halt)** に組み込む停止コマンド

```
casstop /rESSRV01
sleep 5
killmfds.sh
```

クラスタ構成ファイル cmclconf.ascii にてデータ LAN 両系(二重)の定義をしておく

＜検証＞
rp3440-3 でパッケージ ESSRV01 を開始しておく。

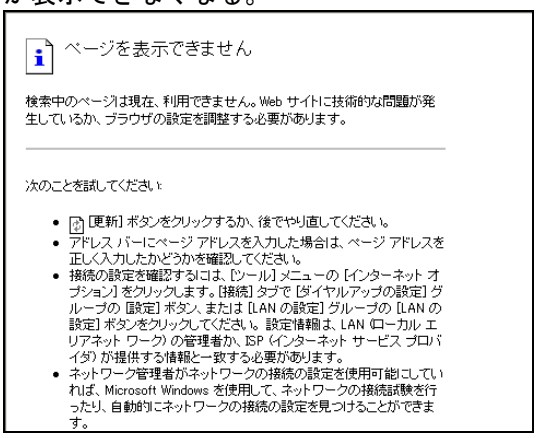


クライアントマシンで Java クライアントプログラムを一定の周期で実行を繰り返す。
シェル例(RunjavaTest.sh)

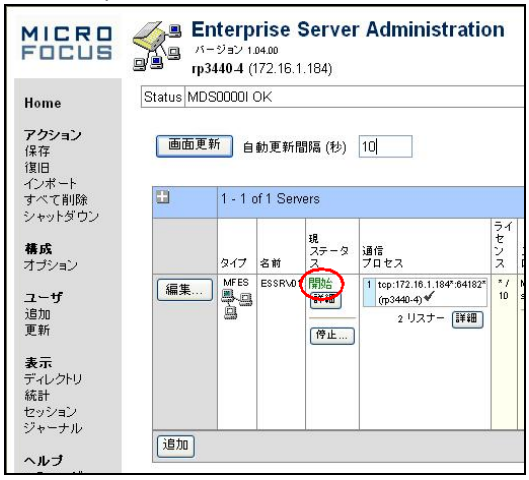
```
#!/bin/sh
while [ "1" == "1" ]; do
java -
classpath .:/home/clinux2/test/mfcobol.jar:/home/clinux2/test/mfconnector.jar:/home/clinux2/test/j2ee.jar clptest
#sleep 10
done
```

稼動側サーバに接続されたデータ LAN のケーブルを 2 本とも引き抜く。

すると、パッケージ ESSRV01 の停止処理により mfds が止まるため rp3440-3 の管理画面が表示できなくなる。



そして rp3440-4 でパッケージ ESSRV01 が開始される。



クライアントプログラム実行結果

```
[root@test]# ./RunjavaTest.sh
06/03/27 12:13:04.81 CBLAPP01 rp3440-3 481
06/03/27 12:13:05.44 CBLAPP01 rp3440-3 482
06/03/27 12:13:06.06 CBLAPP01 rp3440-3 483
06/03/27 12:13:06.72 CBLAPP01 rp3440-3 484
06/03/27 12:13:07.34 CBLAPP01 rp3440-3 485
06/03/27 12:13:07.98 CBLAPP01 rp3440-3 486
06/03/27 12:13:08.63 CBLAPP01 rp3440-3 487
06/03/27 12:13:09.25 CBLAPP01 rp3440-3 488
06/03/27 12:13:09.90 CBLAPP01 rp3440-3 489
06/03/27 12:13:10.54 CBLAPP01 rp3440-3 490
06/03/27 12:13:11.17 CBLAPP01 rp3440-3 491
06/03/27 12:13:11.83 CBLAPP01 rp3440-3 492
javax.resource.spi.ElSSystemException: CobolException Server did not return any data or header executing
srv01.CBLAPP01
java.net.ConnectException: Connection refused (errno:239)
        at java.net.PlainSocketImpl.socketConnect(Native Method)
        at java.net.PlainSocketImpl.doConnect(PlainSocketImpl.java:305)
        at java.net.PlainSocketImpl.connectToAddress(PlainSocketImpl.java:171)
        at java.net.PlainSocketImpl.connect(PlainSocketImpl.java:158)
        at java.net.Socket.connect(Socket.java:462)
        at java.net.Socket.connect(Socket.java:412)
        at java.net.Socket.<init>(Socket.java:308)
        at java.net.Socket.<init>(Socket.java:153)
        at com.microfocus.cobol.connector.transport.BINPStream.createSocketWithTimeout(BINPStream.java:894)
        at com.microfocus.cobol.connector.transport.BINPStream.createSocketWithTimeout(BINPStream.java:825)
        at com.microfocus.cobol.connector.transport.BINPStream.<init>(BINPStream.java:97)
        at com.microfocus.cobol.connector.transport.BINPRemoteCall.open(BINPRemoteCall.java:176)
        at com.microfocus.cobol.connector.transport.BINPRemoteCall.connect(BINPRemoteCall.java:73)
        at com.microfocus.cobol.connector.transport.BINPRemoteCall.reconnect(BINPRemoteCall.java:145)
        at com.microfocus.cobol.connector.PureCobolBean.cobcall(PureCobolBean.java:285)
        at com.microfocus.cobol.connector.cci.CobolInteraction.exec(CobolInteraction.java:247)
        at com.microfocus.cobol.connector.cci.CobolInteraction.execute(CobolInteraction.java:174)
        at clptest.main(clptest.java:48)
javax.resource.spi.ElSSystemException: CobolException Exception throw during open (see getCause() for more
information) (elapsed time=224.0)for 172.16.104.1:9003 (cause class:java.net.ConnectException,
cause:Connection refused (errno:239)) executing srv01.CBLAPP01
        at clptest.main(clptest.java:47)
                        :
                ( 繰り返し・中略 )
                        ;
java.net.ConnectException: Connection refused (errno:239)
        at java.net.PlainSocketImpl.socketConnect(Native Method)
        at java.net.PlainSocketImpl.doConnect(PlainSocketImpl.java:305)
        at java.net.PlainSocketImpl.connectToAddress(PlainSocketImpl.java:171)
        at java.net.PlainSocketImpl.connect(PlainSocketImpl.java:158)
        at java.net.Socket.connect(Socket.java:462)
        at java.net.Socket.connect(Socket.java:412)
        at java.net.Socket.<init>(Socket.java:308)
        at java.net.Socket.<init>(Socket.java:153)
        at com.microfocus.cobol.connector.transport.BINPStream.createSocketWithTimeout(BINPStream.java:894)
        at com.microfocus.cobol.connector.transport.BINPStream.createSocketWithTimeout(BINPStream.java:825)
```

rp3440-3 でサービスが稼動

サービス中断中

```
        at com.microfocus.cobol.connector.transport.BINPStream.<init>(BINPStream.java:97)
        at com.microfocus.cobol.connector.transport.BINPRemoteCall.open(BINPRemoteCall.java:176)
        at com.microfocus.cobol.connector.transport.BINPRemoteCall.connect(BINPRemoteCall.java:73)
        at com.microfocus.cobol.connector.transport.BINPRemoteCall.reconnect(BINPRemoteCall.java:145)
        at com.microfocus.cobol.connector.PureCobolBean.cobcall(PureCobolBean.java:285)
        at com.microfocus.cobol.connector.cci.CobolInteraction.exec(CobolInteraction.java:247)
        at com.microfocus.cobol.connector.cci.CobolInteraction.execute(CobolInteraction.java:174)
        at clptest.main(clptest.java:48)
javax.resource.spi.EISSystemException: CobolException Exception throw during open (see getCause() for more
information) (elapsed time=230.0)for 172.16.104.1:9003 (cause class:java.net.ConnectException,
cause:Connection refused (errno:239)) executing srv01.CBLAPP01
06/03/27 12:13:29.88 CBLAPP01 rp3440-4 493
06/03/27 12:13:30.54 CBLAPP01 rp3440-4 494
06/03/27 12:13:31.19 CBLAPP01 rp3440-4 495
06/03/27 12:13:31.85 CBLAPP01 rp3440-4 496
06/03/27 12:13:32.49 CBLAPP01 rp3440-4 497
06/03/27 12:13:33.13 CBLAPP01 rp3440-4 498
06/03/27 12:13:33.77 CBLAPP01 rp3440-4 499
06/03/27 12:13:34.46 CBLAPP01 rp3440-4 500
06/03/27 12:13:35.09 CBLAPP01 rp3440-4 501
06/03/27 12:13:35.70 CBLAPP01 rp3440-4 502
06/03/27 12:13:36.38 CBLAPP01 rp3440-4 503
06/03/27 12:13:36.97 CBLAPP01 rp3440-4 504
06/03/27 12:13:37.61 CBLAPP01 rp3440-4 505
06/03/27 12:13:38.25 CBLAPP01 rp3440-4 506
06/03/27 12:13:38.89 CBLAPP01 rp3440-4 507
06/03/27 12:13:39.54 CBLAPP01 rp3440-4 508

[root@test]#
```

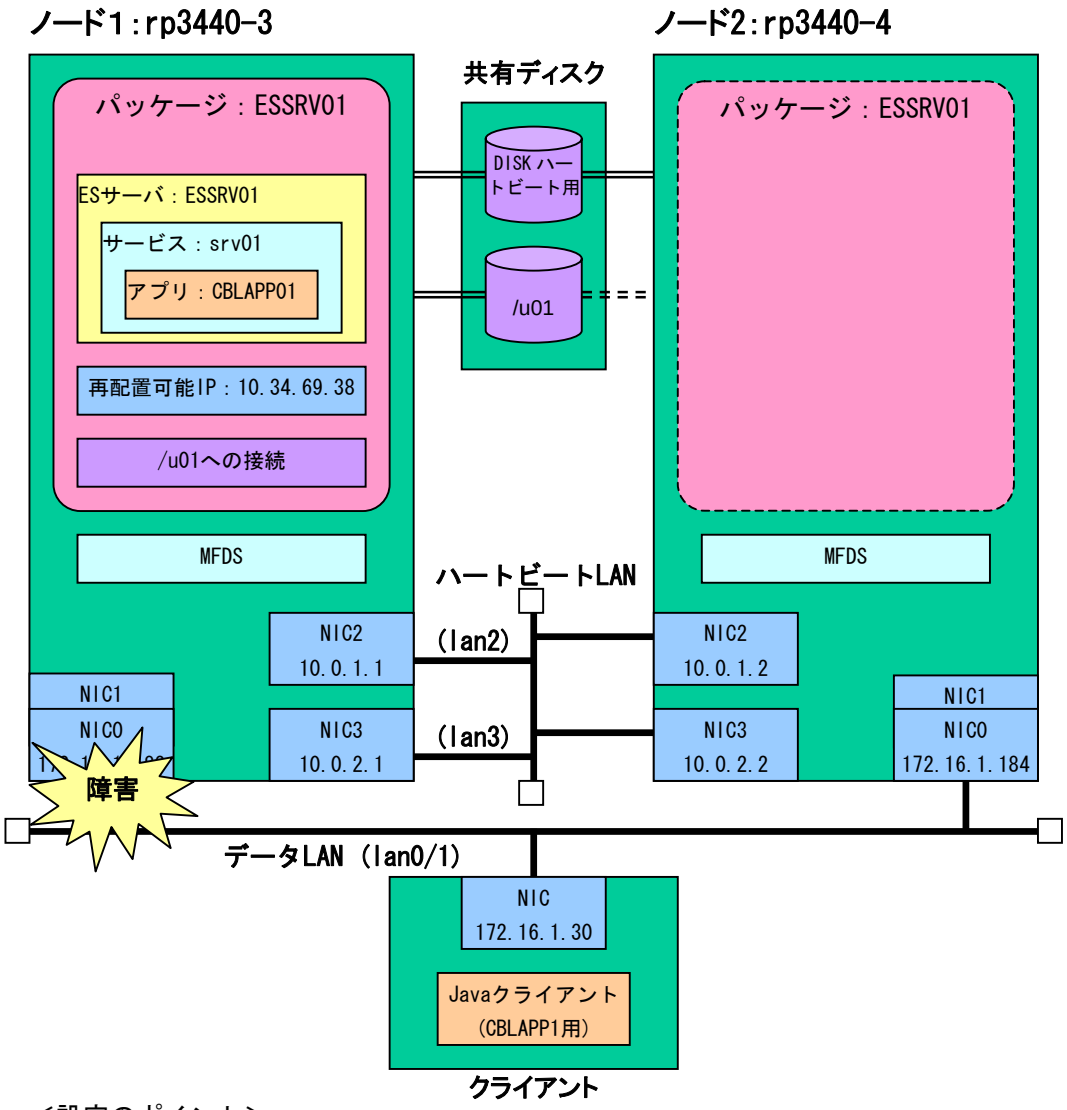
rp3440-4 でサービスを引継ぎ。
データファイルの連番も継続

＜結果＞
待機側のノードで Enterprise Server の関連プロセスは起動しない状態からのパッケージの移動を確認。
また、一定時間のサービス中断後、同じサービスを待機側で提供できていることも確認。

パターン⑤ MFDS 起動で待機

＜目的＞
待機側のノードで Enterprise Server の基本プロセス MFDS のみ起動した状態からのフェイルオーバーを確認する。
フェイルオーバー前と後で同じサービスが継続される事も確認する。

＜構成＞



＜設定のポイント＞
それぞれ各ノードであらかじめ MFDS を立ち上げておく。
MFDS の制御はパッケージに含めない。
ES 管理シェル mf.sh の **start)** に組み込む起動コマンド

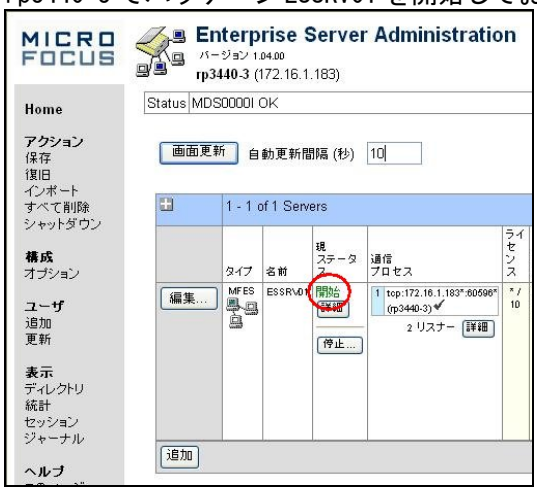
```
killallcas.sh  ESSRV01
casstart /rESSRV01
```

ES 管理シェル mf.sh の **halt)** に組み込む停止コマンド

```
casstop /rESSRV01
```

クラスタ構成ファイル cmclconf.ascii にてデータ LAN 両系(二重)の定義をしておく

＜検証＞
rp3440-3 でパッケージ ESSRV01 を開始しておく。



rp3440-4 では ES サーバ(パッケージ ESSRV01)は停止している。

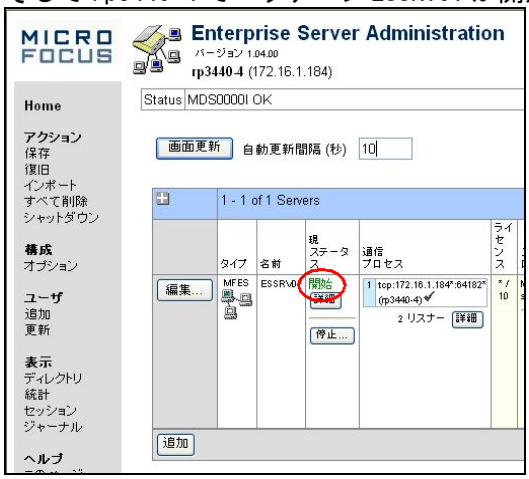
クライアントマシンで Java クライアントプログラムを一定の周期で実行を繰り返す。
シェル例(RunjavaTest.sh)

```
#!/bin/sh
while [ "1" == "1" ]; do
java -
classpath .:/home/clinux2/test/mfcobol.jar:/home/clinux2/test/mfconnector.jar:/home/clinux2/t
est/j2ee.jar clptest
#sleep 10
done
```

稼動側サーバに接続されたデータ LAN のケーブルを 2 本とも引き抜く。

すると、パッケージ ESSRV01 の停止処理により ES サーバ ESSRV01 が停止する。
注）タイミングによって ES サーバ ESSRV01 のステータスは「応答なし」なることがあります。

そして rp3440-4 でパッケージ ESSRV01 が開始される。



クライアントプログラム実行結果

```
[root@test]# ./RunjavaTest.sh
06/03/27 12:27:31.65 CBLAPP01 rp3440-3 548
06/03/27 12:27:32.28 CBLAPP01 rp3440-3 549
06/03/27 12:27:32.91 CBLAPP01 rp3440-3 550
06/03/27 12:27:33.55 CBLAPP01 rp3440-3 551
06/03/27 12:27:34.18 CBLAPP01 rp3440-3 552
06/03/27 12:27:34.86 CBLAPP01 rp3440-3 553
06/03/27 12:27:35.50 CBLAPP01 rp3440-3 554
06/03/27 12:27:36.12 CBLAPP01 rp3440-3 555
javax.resource.spi.EISSystemException: CobolException Server did not return any data or header executing
srv01.CBLAPP01
java.net.ConnectException: Connection refused (errno:239)
        at java.net.PlainSocketImpl.socketConnect(Native Method)
        at java.net.PlainSocketImpl.doConnect(PlainSocketImpl.java:305)
        at java.net.PlainSocketImpl.connectToAddress(PlainSocketImpl.java:171)
        at java.net.PlainSocketImpl.connect(PlainSocketImpl.java:158)
        at java.net.Socket.connect(Socket.java:462)
        at java.net.Socket.connect(Socket.java:412)
        at java.net.Socket.<init>(Socket.java:308)
        at java.net.Socket.<init>(Socket.java:153)
        at com.microfocus.cobol.connector.transport.BINPStream.createSocketWithTimeout(BINPStream.java:894)
        at com.microfocus.cobol.connector.transport.BINPStream.createSocketWithTimeout(BINPStream.java:825)
        at com.microfocus.cobol.connector.transport.BINPStream.<init>(BINPStream.java:97)
        at com.microfocus.cobol.connector.transport.BINPRemoteCall.open(BINPRemoteCall.java:176)
        at com.microfocus.cobol.connector.transport.BINPRemoteCall.connect(BINPRemoteCall.java:73)
        at com.microfocus.cobol.connector.transport.BINPRemoteCall.reconnect(BINPRemoteCall.java:145)
        at com.microfocus.cobol.connector.PureCobolBean.cobcall(PureCobolBean.java:285)
        at com.microfocus.cobol.connector.cci.CobolInteraction.exec(CobolInteraction.java:247)
        at com.microfocus.cobol.connector.cci.CobolInteraction.execute(CobolInteraction.java:174)
        at clptest.main(clptest.java:48)
javax.resource.spi.EISSystemException: CobolException Exception throw during open (see getCause() for more
information) (elapsed time=222.0)for 172.16.104.1:9003 (cause class:java.net.ConnectException,
cause:Connection refused (errno:239)) executing srv01.CBLAPP01
                        :
                  (  繰り返し・中略  )
                        ;
```

rp3440-3 でサービスが稼動

サービス中断中

```
java.net.ConnectException: Connection refused (errno:239)
        at java.net.PlainSocketImpl.socketConnect(Native Method)
        at java.net.PlainSocketImpl.doConnect(PlainSocketImpl.java:305)
        at java.net.PlainSocketImpl.connectToAddress(PlainSocketImpl.java:171)
        at java.net.PlainSocketImpl.connect(PlainSocketImpl.java:158)
        at java.net.Socket.connect(Socket.java:462)
        at java.net.Socket.connect(Socket.java:412)
        at java.net.Socket.<init>(Socket.java:308)
        at java.net.Socket.<init>(Socket.java:153)
        at com.microfocus.cobol.connector.transport.BINPStream.createSocketWithTimeout(BINPStream.java:894)
        at com.microfocus.cobol.connector.transport.BINPStream.createSocketWithTimeout(BINPStream.java:825)
        at com.microfocus.cobol.connector.transport.BINPStream.<init>(BINPStream.java:97)
        at com.microfocus.cobol.connector.transport.BINPRemoteCall.open(BINPRemoteCall.java:176)
        at com.microfocus.cobol.connector.transport.BINPRemoteCall.connect(BINPRemoteCall.java:73)
        at com.microfocus.cobol.connector.transport.BINPRemoteCall.reconnect(BINPRemoteCall.java:145)
        at com.microfocus.cobol.connector.PureCobolBean.cobcall(PureCobolBean.java:285)
        at com.microfocus.cobol.connector.cci.CobolInteraction.exec(CobolInteraction.java:247)
        at com.microfocus.cobol.connector.cci.CobolInteraction.execute(CobolInteraction.java:174)
        at clptest.main(clptest.java:48)
javax.resource.spi.EISSystemException: CobolException Exception throw during open (see getCause() for more
information) (elapsed time=220.0)for 172.16.104.1:9003 (cause class:java.net.ConnectException,
cause:Connection refused (errno:239)) executing srv01.CBLAPP01
06/03/27 12:27:54.12 CBLAPP01 rp3440-4 556
06/03/27 12:27:54.80 CBLAPP01 rp3440-4 557
06/03/27 12:27:55.65 CBLAPP01 rp3440-4 558
06/03/27 12:27:56.27 CBLAPP01 rp3440-4 559
06/03/27 12:27:56.91 CBLAPP01 rp3440-4 560
06/03/27 12:27:57.56 CBLAPP01 rp3440-4 561
06/03/27 12:27:58.20 CBLAPP01 rp3440-4 562
06/03/27 12:27:58.84 CBLAPP01 rp3440-4 563
06/03/27 12:27:59.47 CBLAPP01 rp3440-4 564
06/03/27 12:28:00.09 CBLAPP01 rp3440-4 565
06/03/27 12:28:00.74 CBLAPP01 rp3440-4 566
06/03/27 12:28:01.39 CBLAPP01 rp3440-4 567

[root@test]#
```

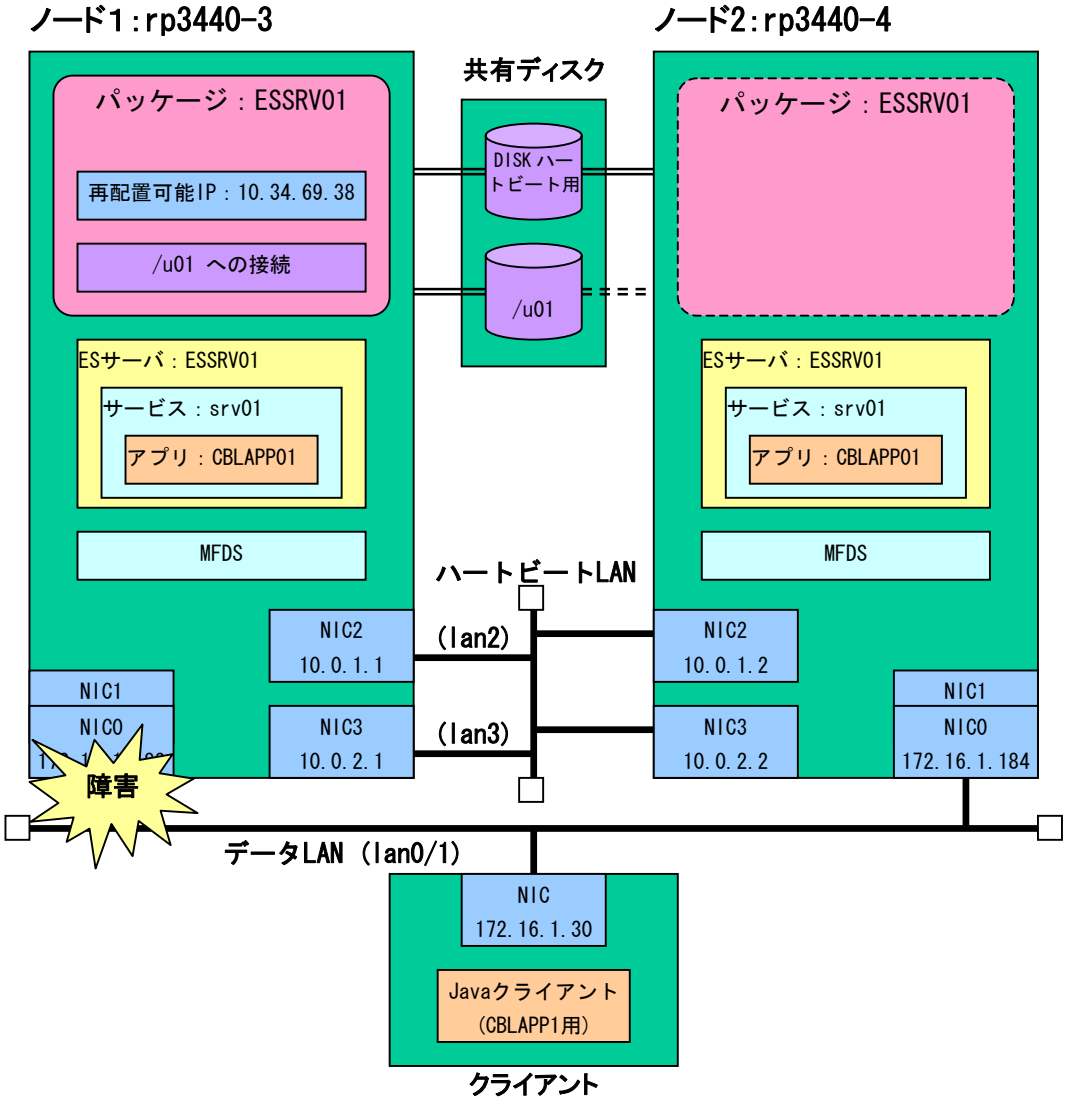
rp3440-4 でサービスを引継ぎ。
データファイルの連番も継続

＜結果＞
待機側のノードで Enterprise Server の基本プロセス MFDS のみ起動した状態からのパッケージの移動を確認。
また、一定時間のサービス中断後、同じサービスを待機側で提供できていることも確認。

パターン⑤ ES サーバ起動で待機

＜目的＞
待機側のノードで Enterprise Server の基本プロセス MFDS と ES サーバを起動した状態からのフェイルオーバーを確認する。
フェイルオーバー前と後で同じサービスが継続される事も確認する。

＜構成＞

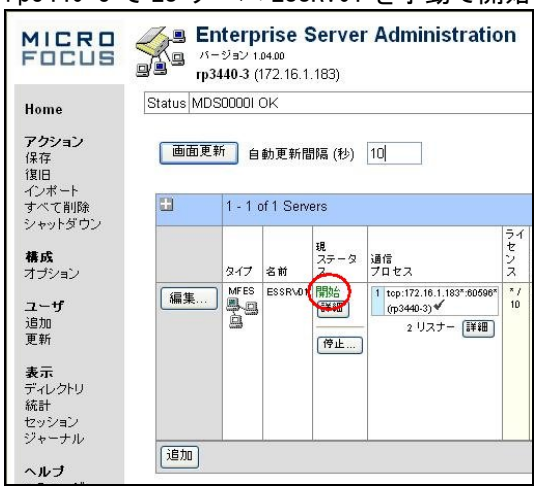


＜設定のポイント＞

それぞれ各ノードであらかじめ MFDS と ES サーバ ESSRV01 を立ち上げておく。
パッケージでは再配置可能 IP アドレスと共有ディスクの切り替えのみを行なう。

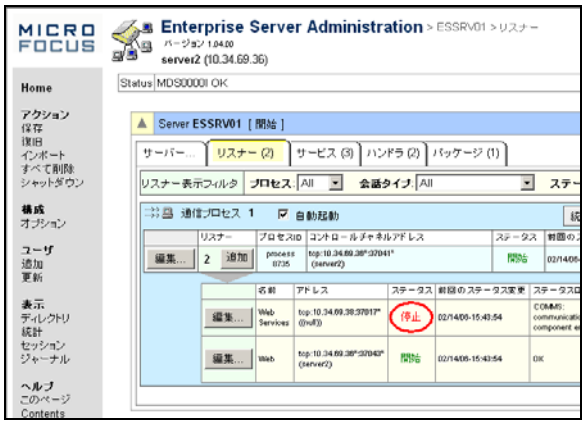
クラスタ構成ファイル cmclconf.ascii にてデータ LAN 両系(二重)の定義をしておく

＜検証＞
rp3440-3 で ES サーバ ESSRV01 を手動で開始しておく。



rp3440-4 で同様に ES サーバ ESSRV01 を手動で開始する。

ところが、rp3440-4 でESSRV01 を立ち上げるとリスナーがエラーで立ち上がらない。

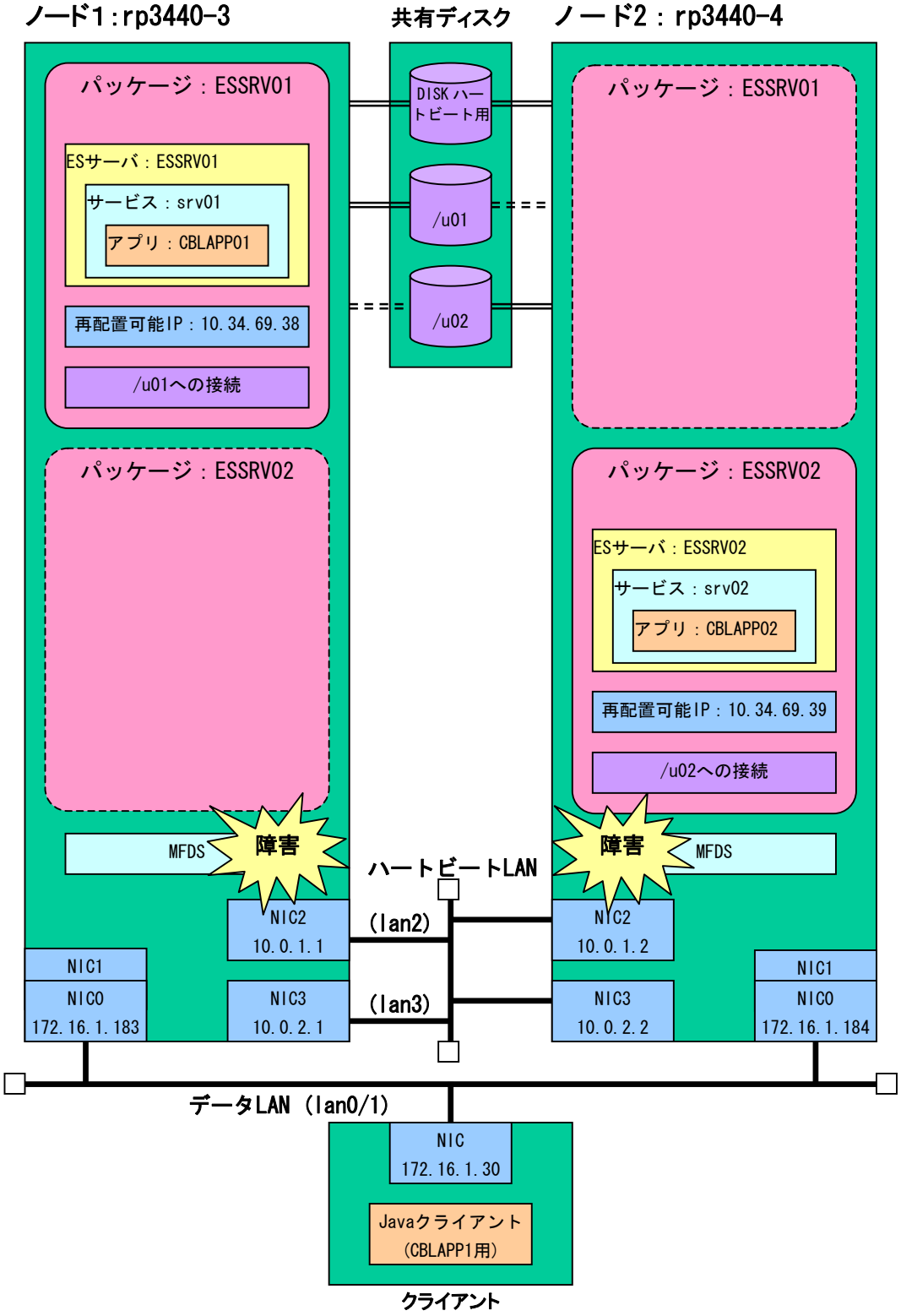


理由：Enterprise Server のリスナーは指定した IP アドレスが有効になっている必要がある。

＜結果＞
Enterprise Server の制限事項により、待機側の ES サーバを正常に立ち上がらせることができないため、本パターンの前提条件を満たせなかった。
よって本パターンでの動作検証は不可。

パターン⑦ 別の業務を提供しながら待機

＜目的＞
待機側のノードで Enterprise Server の MFDS を起動し、別の ES サーバによるサービスを起動した状態からのフェイルオーバーを確認する。
フェイルオーバー前と後で同じサービスが継続される事も確認する。
＜構成＞



＜設定のポイント＞
MFDS の制御はパッケージに含めない。

ESSRV01 の開始シェルに組み込む起動コマンド

```
killallcas.sh ESSRV01
casstart /rESSRV01
```

ESSRV01 の終了シェルに組み込む停止コマンド

```
casstop /rESSRV01
```

ESSRV02 の開始シェルに組み込む起動コマンド

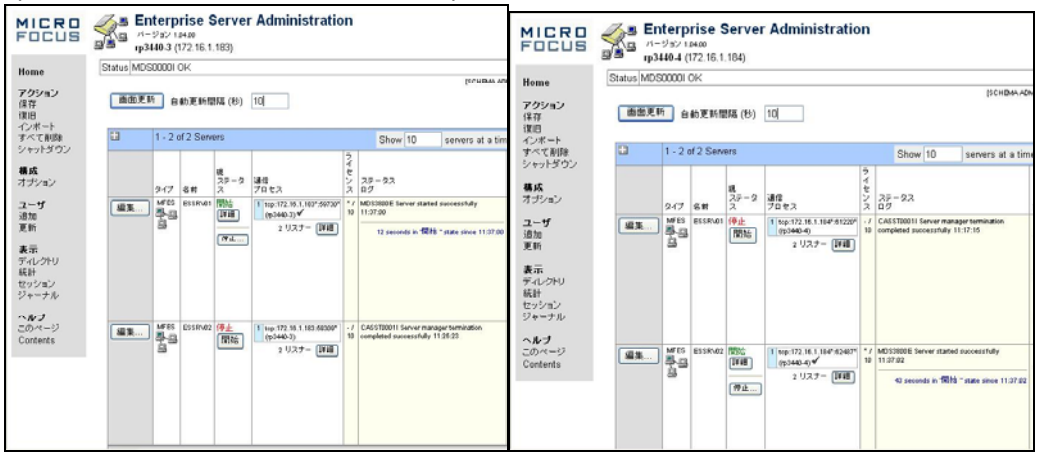
```
killallcas.sh ESSRV02
casstart /rESSRV02
```

ESSRV02 の終了シェルに組み込む停止コマンド

```
casstop /rESSRV02
```

cntl ファイルにて監視用のシェルを起動するように設定

＜検証＞
それぞれ各ノードであらかじめ MFDS を立ち上げる。
rp3440-3 でパッケージ ESSRV01 を、rp3440-4 でパッケージ ESSRV02 を開始しておく。



クライアントマシンで各サービスに対する Java クライアントプログラムを一定の周期で実行を繰り返す。
シェル例(RunjavaTest.sh)

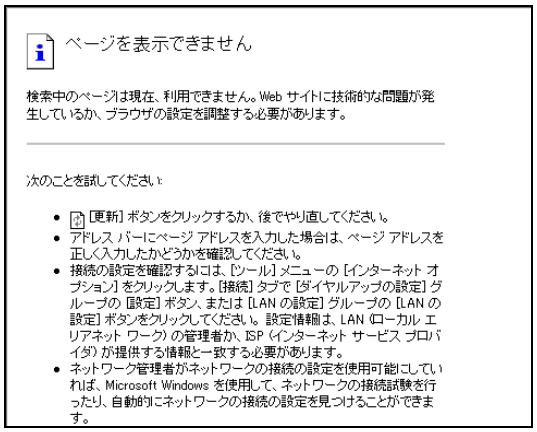
```
#!/bin/sh
while [ "1" == "1" ]; do
java -
classpath .:/home/clinux2/test/mfcobol.jar:/home/clinux2/test/mfconnector.jar:/home/clinux2/t
est/j2ee.jar clptest
#sleep 10
done
```

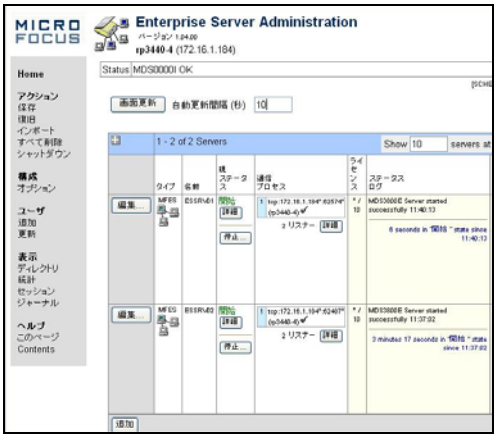
次に MFDS のプロセスを kill する。

- kill -9 <mfds32 の pid>

すると、ES 管理シェル mf.sh の **monitor**)が障害を検知しフェイルオーバーが開始する。

MFDS が止まるため rp3440-3 の管理画面が表示できなくなる。
また rp3440-4 にパッケージが移動する。





クライアントプログラム実行結果(CBLAPP01)

```
[root@test]#./RunjavaTest.sh
06/03/27 11:06:17.34 CBLAPP01 rp3440-3 186
06/03/27 11:06:18.07 CBLAPP01 rp3440-3 187
06/03/27 11:06:18.72 CBLAPP01 rp3440-3 188
06/03/27 11:06:19.36 CBLAPP01 rp3440-3 189
06/03/27 11:06:20.01 CBLAPP01 rp3440-3 190
06/03/27 11:06:20.63 CBLAPP01 rp3440-3 191
06/03/27 11:06:21.23 CBLAPP01 rp3440-3 192
06/03/27 11:06:21.85 CBLAPP01 rp3440-3 193
06/03/27 11:06:22.44 CBLAPP01 rp3440-3 194          rp3440-3 でサービスが稼動
06/03/27 11:06:23.12 CBLAPP01 rp3440-3 195
06/03/27 11:06:23.72 CBLAPP01 rp3440-3 196
06/03/27 11:06:24.39 CBLAPP01 rp3440-3 197
06/03/27 11:06:25.01 CBLAPP01 rp3440-3 198
06/03/27 11:06:25.67 CBLAPP01 rp3440-3 199
06/03/27 11:06:26.29 CBLAPP01 rp3440-3 200
06/03/27 11:06:27.01 CBLAPP01 rp3440-3 201
java.net.ConnectException: Connection refused (errno:239)
        at java.net.PlainSocketImpl.socketConnect(Native Method)
        at java.net.PlainSocketImpl.doConnect(PlainSocketImpl.java:305)
        at java.net.PlainSocketImpl.connectToAddress(PlainSocketImpl.java:171)        サービス中断中
        at java.net.Socket.connect(Socket.java:452)
        at java.net.PlainSocketImpl.connect(PlainSocketImpl.java:158)
        at java.net.Socket.connect(Socket.java:462)
        at java.net.Socket.connect(Socket.java:412)
        at java.net.Socket.<init>(Socket.java:308)
        at java.net.Socket.<init>(Socket.java:153)
        at com.microfocus.cobol.connector.transport.BINPStream.createSocketWithTimeout(BINPStream.java:894)
        at com.microfocus.cobol.connector.transport.BINPStream.createSocketWithTimeout(BINPStream.java:825)
        at com.microfocus.cobol.connector.transport.BINPStream.<init>(BINPStream.java:97)
        at com.microfocus.cobol.connector.transport.BINPRemoteCall.open(BINPRemoteCall.java:176)
        at com.microfocus.cobol.connector.transport.BINPRemoteCall.connect(BINPRemoteCall.java:73)
        at com.microfocus.cobol.connector.transport.BINPRemoteCall.reconnect(BINPRemoteCall.java:145)
        at com.microfocus.cobol.connector.PureCobolBean.cobcall(PureCobolBean.java:285)
        at com.microfocus.cobol.connector.cci.CobolInteraction.exec(CobolInteraction.java:247)
        at com.microfocus.cobol.connector.cci.CobolInteraction.execute(CobolInteraction.java:174)
        at clptest.main(clptest.java:48)
javax.resource.spi.EISSystemException: CobolException Exception throw during open (see getCause() for more
information) (elapsed time=225.0)for 172.16.104.1:9003 (cause class:java.net.ConnectException,
cause:Connection refused (errno:239)) executing srv01.CBLAPP01
                        :
                ( 繰り返し・中略 )
                        ;
java.net.ConnectException: Connection refused (errno:239)
        at java.net.PlainSocketImpl.socketConnect(Native Method)
        at java.net.PlainSocketImpl.doConnect(PlainSocketImpl.java:305)
        at java.net.PlainSocketImpl.connectToAddress(PlainSocketImpl.java:171)
        at java.net.PlainSocketImpl.connect(PlainSocketImpl.java:158)
        at java.net.Socket.connect(Socket.java:462)
        at java.net.Socket.connect(Socket.java:412)
```

```
        at java.net.Socket.<init>(Socket.java:308)
        at java.net.Socket.<init>(Socket.java:153)
        at com.microfocus.cobol.connector.transport.BINPStream.createSocketWithTimeout(BINPStream.java:894)
        at com.microfocus.cobol.connector.transport.BINPStream.createSocketWithTimeout(BINPStream.java:825)
        at com.microfocus.cobol.connector.transport.BINPStream.<init>(BINPStream.java:97)
        at com.microfocus.cobol.connector.transport.BINPRemoteCall.open(BINPRemoteCall.java:176)
        at com.microfocus.cobol.connector.transport.BINPRemoteCall.connect(BINPRemoteCall.java:73)
        at com.microfocus.cobol.connector.transport.BINPRemoteCall.reconnect(BINPRemoteCall.java:145)
        at com.microfocus.cobol.connector.PureCobolBean.cobcall(PureCobolBean.java:285)
        at com.microfocus.cobol.connector.cci.CobolInteraction.exec(CobolInteraction.java:247)
        at com.microfocus.cobol.connector.cci.CobolInteraction.execute(CobolInteraction.java:174)
        at clptest.main(clptest.java:48)
javax.resource.spi.EISSystemException: CobolException Exception throw during open (see getCause() for more
information) (elapsed time=224.0)for 172.16.104.1:9003 (cause class:java.net.ConnectException,
cause:Connection refused (errno:239)) executing srv01.CBLAPP01
06/03/27 11:07:02.16 CBLAPP01 rp3440-4 202
06/03/27 11:07:02.81 CBLAPP01 rp3440-4 203
06/03/27 11:07:03.45 CBLAPP01 rp3440-4 204
06/03/27 11:07:04.09 CBLAPP01 rp3440-4 205
06/03/27 11:07:04.92 CBLAPP01 rp3440-4 206
06/03/27 11:07:05.57 CBLAPP01 rp3440-4 207
06/03/27 11:07:06.18 CBLAPP01 rp3440-4 208
06/03/27 11:07:06.85 CBLAPP01 rp3440-4 209
06/03/27 11:07:07.49 CBLAPP01 rp3440-4 210
06/03/27 11:07:08.09 CBLAPP01 rp3440-4 211
06/03/27 11:07:08.76 CBLAPP01 rp3440-4 212
06/03/27 11:07:09.40 CBLAPP01 rp3440-4 213
06/03/27 11:07:10.01 CBLAPP01 rp3440-4 214
06/03/27 11:07:10.65 CBLAPP01 rp3440-4 215
06/03/27 11:07:11.27 CBLAPP01 rp3440-4 216
06/03/27 11:07:11.96 CBLAPP01 rp3440-4 217
06/03/27 11:07:12.74 CBLAPP01 rp3440-4 218
06/03/27 11:07:13.48 CBLAPP01 rp3440-4 219
06/03/27 11:07:14.23 CBLAPP01 rp3440-4 220
06/03/27 11:07:14.87 CBLAPP01 rp3440-4 221
06/03/27 11:07:15.53 CBLAPP01 rp3440-4 222
06/03/27 11:07:16.39 CBLAPP01 rp3440-4 223
06/03/27 11:07:17.03 CBLAPP01 rp3440-4 224
06/03/27 11:07:17.68 CBLAPP01 rp3440-4 225

[root@test]#
```

server2 でサービスを引継ぎ。
データファイルの連番も継続

クライアントプログラム実行結果(CBLAPP02)

```
[root@test]# ./RunjavaTest2.sh
06/03/27 11:06:18.80 CBLAPP02 rp3440-4 348
06/03/27 11:06:19.43 CBLAPP02 rp3440-4 349
06/03/27 11:06:20.05 CBLAPP02 rp3440-4 350
06/03/27 11:06:20.68 CBLAPP02 rp3440-4 351
06/03/27 11:06:21.40 CBLAPP02 rp3440-4 352
06/03/27 11:06:22.05 CBLAPP02 rp3440-4 353
06/03/27 11:06:22.67 CBLAPP02 rp3440-4 354
06/03/27 11:06:23.28 CBLAPP02 rp3440-4 355
06/03/27 11:06:23.88 CBLAPP02 rp3440-4 356
06/03/27 11:06:24.50 CBLAPP02 rp3440-4 357
06/03/27 11:06:25.13 CBLAPP02 rp3440-4 358
06/03/27 11:06:25.73 CBLAPP02 rp3440-4 359
06/03/27 11:06:26.37 CBLAPP02 rp3440-4 360
06/03/27 11:06:27.06 CBLAPP02 rp3440-4 361
06/03/27 11:06:27.69 CBLAPP02 rp3440-4 362
06/03/27 11:06:28.30 CBLAPP02 rp3440-4 363
06/03/27 11:06:28.95 CBLAPP02 rp3440-4 364
06/03/27 11:06:29.60 CBLAPP02 rp3440-4 365
06/03/27 11:06:30.23 CBLAPP02 rp3440-4 366
06/03/27 11:06:30.84 CBLAPP02 rp3440-4 367
06/03/27 11:06:31.50 CBLAPP02 rp3440-4 368
06/03/27 11:06:32.14 CBLAPP02 rp3440-4 369
06/03/27 11:06:32.77 CBLAPP02 rp3440-4 370
06/03/27 11:06:33.42 CBLAPP02 rp3440-4 371
06/03/27 11:06:34.04 CBLAPP02 rp3440-4 372
06/03/27 11:06:34.63 CBLAPP02 rp3440-4 373
06/03/27 11:06:35.27 CBLAPP02 rp3440-4 374
06/03/27 11:06:35.94 CBLAPP02 rp3440-4 375
06/03/27 11:06:36.77 CBLAPP02 rp3440-4 376
```

server2 でサービス中断せず

```
06/03/27 11:06:37.40 CBLAPP02 rp3440-4 377
06/03/27 11:06:37.99 CBLAPP02 rp3440-4 378
06/03/27 11:06:38.68 CBLAPP02 rp3440-4 379
06/03/27 11:06:39.33 CBLAPP02 rp3440-4 380
06/03/27 11:06:39.94 CBLAPP02 rp3440-4 381
06/03/27 11:06:40.59 CBLAPP02 rp3440-4 382
06/03/27 11:06:41.22 CBLAPP02 rp3440-4 383
06/03/27 11:06:41.85 CBLAPP02 rp3440-4 384
06/03/27 11:06:42.46 CBLAPP02 rp3440-4 385
06/03/27 11:06:43.16 CBLAPP02 rp3440-4 386
06/03/27 11:06:43.85 CBLAPP02 rp3440-4 387
06/03/27 11:06:44.46 CBLAPP02 rp3440-4 388
06/03/27 11:06:45.09 CBLAPP02 rp3440-4 389
06/03/27 11:06:45.85 CBLAPP02 rp3440-4 390
06/03/27 11:06:46.61 CBLAPP02 rp3440-4 391
06/03/27 11:06:47.22 CBLAPP02 rp3440-4 392
06/03/27 11:06:47.91 CBLAPP02 rp3440-4 393
06/03/27 11:06:48.55 CBLAPP02 rp3440-4 394
06/03/27 11:06:49.16 CBLAPP02 rp3440-4 395
06/03/27 11:06:49.76 CBLAPP02 rp3440-4 396
06/03/27 11:06:50.40 CBLAPP02 rp3440-4 397
06/03/27 11:06:51.09 CBLAPP02 rp3440-4 398
06/03/27 11:06:51.71 CBLAPP02 rp3440-4 399
06/03/27 11:06:52.34 CBLAPP02 rp3440-4 400
06/03/27 11:06:52.94 CBLAPP02 rp3440-4 401
06/03/27 11:06:53.54 CBLAPP02 rp3440-4 402
06/03/27 11:06:54.20 CBLAPP02 rp3440-4 403
06/03/27 11:06:54.97 CBLAPP02 rp3440-4 404
06/03/27 11:06:55.69 CBLAPP02 rp3440-4 405
06/03/27 11:06:56.28 CBLAPP02 rp3440-4 406
06/03/27 11:06:56.89 CBLAPP02 rp3440-4 407
06/03/27 11:06:57.56 CBLAPP02 rp3440-4 408
06/03/27 11:06:58.18 CBLAPP02 rp3440-4 409
06/03/27 11:06:58.93 CBLAPP02 rp3440-4 410
06/03/27 11:06:59.63 CBLAPP02 rp3440-4 411
06/03/27 11:07:00.35 CBLAPP02 rp3440-4 412
06/03/27 11:07:00.99 CBLAPP02 rp3440-4 413
06/03/27 11:07:01.75 CBLAPP02 rp3440-4 414
06/03/27 11:07:02.48 CBLAPP02 rp3440-4 415
```

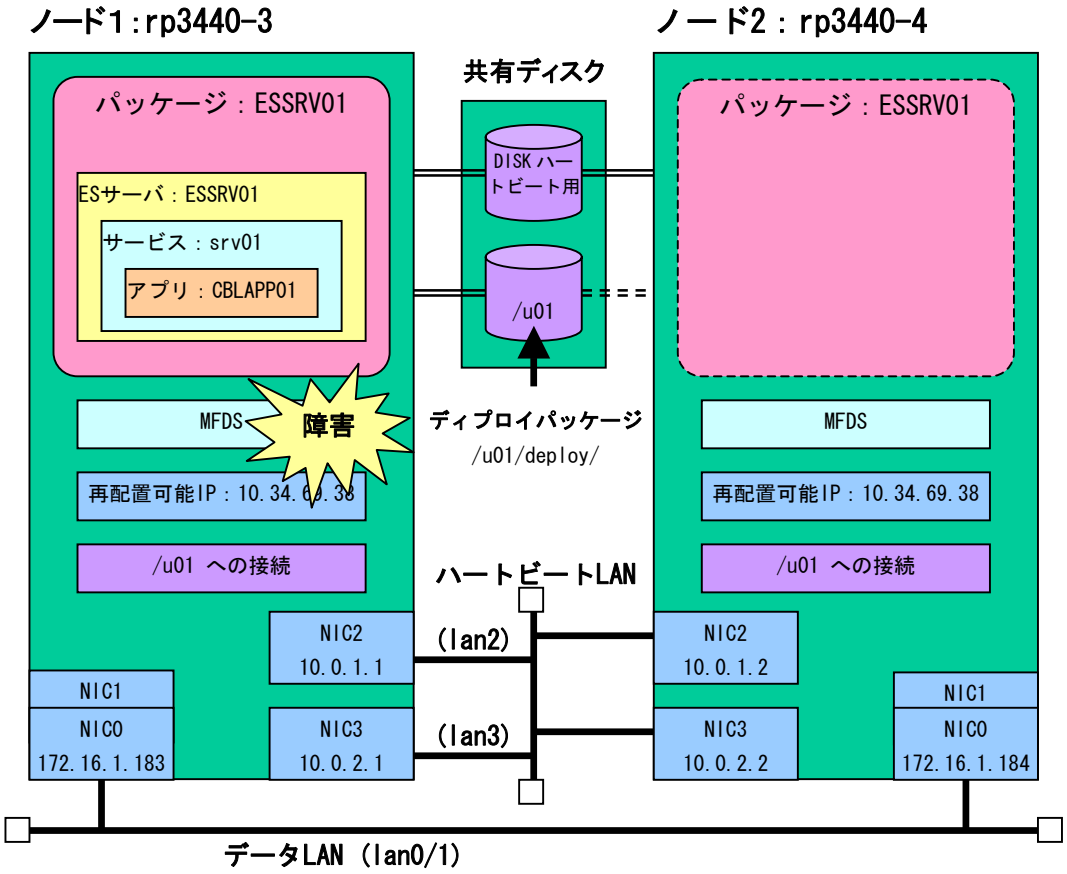
同様に同じ状態から rp3440-4 の MFDS を直接手動で停止し動作を確認する。

＜結果＞
フェイルオーバーさせる側のノードで Enterprise Server の MFDS 起動した状態かつ別の ES サーバによるサービスを継続提供したままの状態で、パッケージの移動が行える事を双方向で確認。

パターン⑧ ディプロイパッケージの共有

＜目的＞
ディプロイリソースの一元管理を目的として、Enterprise Server にディプロイするパッケージを共有ファイルに配置し、フェイルオーバーを確認する。

＜構成＞



＜設定のポイント＞
ES 管理シェル mf.sh の start) に組み込む起動コマンド
※ パッケージのディプロイのコマンド mfdepinst は運用上の必要に応じて組み込む。

```
killallcas.sh ESSRV01
cd /u01/deploy/srv01
mfdepinst srv01.car
casstart /rESSRV01
```

ES 管理シェル mf.sh の halt) に組み込む停止コマンド
※ アンディプロイの処理は手動前提となる。

```
casstop /rESSRV01
killmfds.sh
```

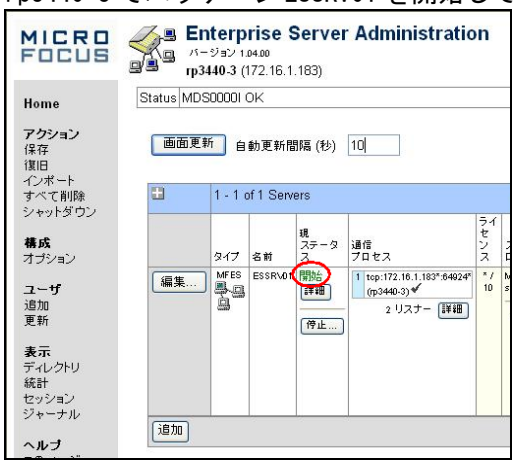
cntl ファイルにて監視用のシェルを起動するように設定

ES サーバ ESSRV01 にディプロイするサービスは一度削除し、共有ディスクに配置したディプロイパッケージ(.car ファイル)から再度ディプロイ作業を行なう。

本検証では/u01/deploy にディプロイパッケージを配置した

この作業を両ノード共に実施する。
　注）起動時に mfdepinst コマンドにてディプロイする場合は不要

＜検証＞
rp3440-3 でパッケージ ESSRV01 を開始しておく。



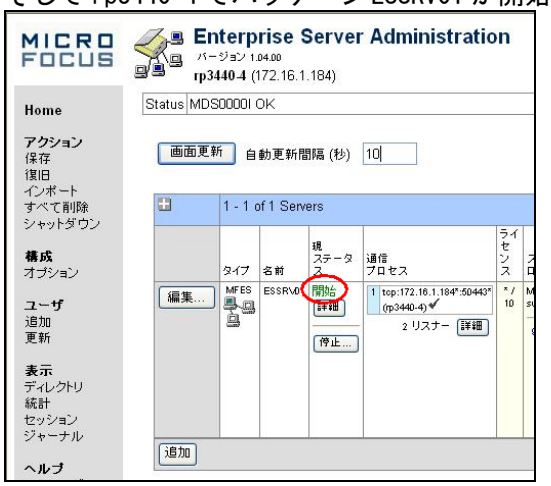
クライアントマシンで Java クライアントプログラムを実行し動作を確認する。

次に rp3440-3 の MFDS のプロセスを kill する。

- kill -9 <mfds32 の pid>

すると、ES 管理シェル mf.sh の **monitor**)が障害を検知しフェイルオーバーが開始する。

そして rp3440-4 でパッケージ ESSRV01 が開始される。



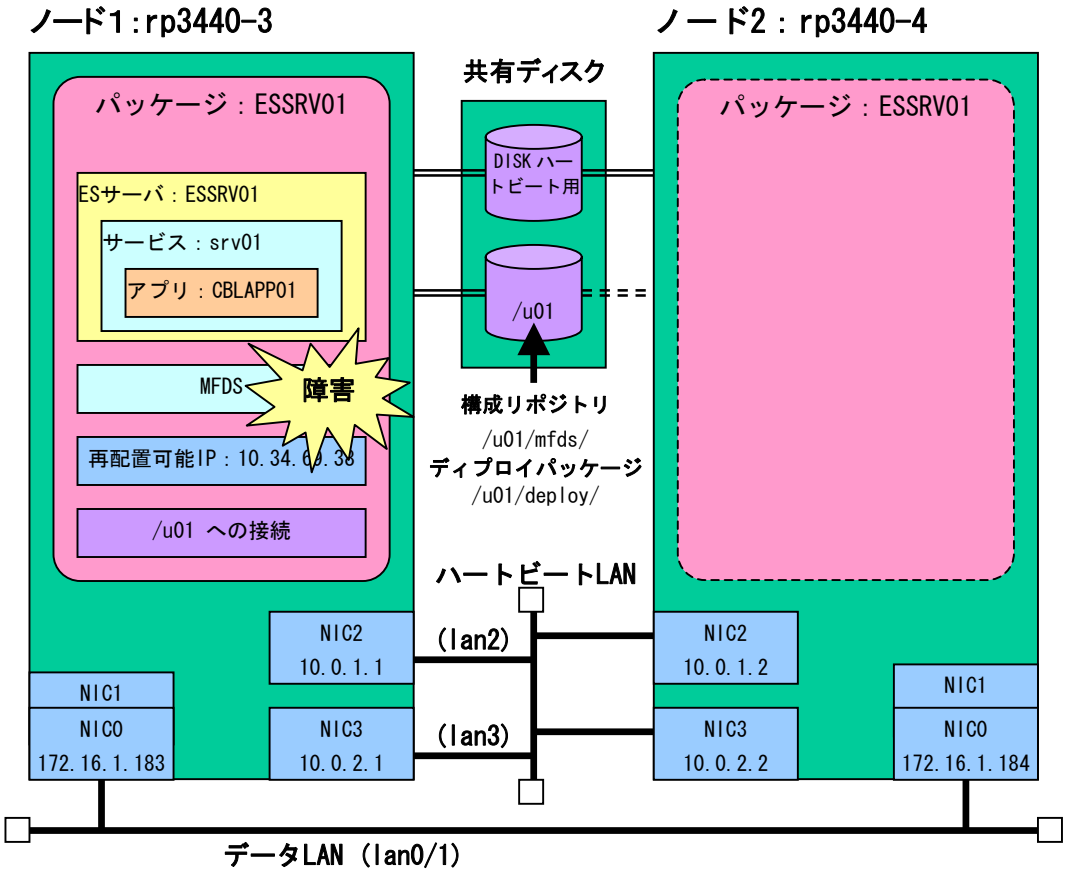
そこでもう一度、クライアントマシンで Java クライアントプログラムを実行し動作を確認する。

＜結果＞
共有ディスクにディプロイファイルを配置して２つのノード間でディプロイパッケージを共有する事は可能。

パターン⑨ 構成リポジトリの共有

＜目的＞
Enterprise Server の各種設定情報を格納した構成リポジトリを共有ファイルに配置し、フェイルオーバーでこの設定情報が引き継がれるかを確認する。

＜構成＞



＜設定のポイント＞
構成リポジトリは MFDS 単位で一箇所のみ有効になり、MFDS の起動時から常に利用され続ける。よってパッケージの切り替え時には MFDS の制御が必須となる。

ES 管理シェル mf.sh の start) に組み込む起動コマンド

```
killmfds.sh
killallcas.sh ESSRV01
mfds &
casstart /rESSRV01
```

ES 管理シェル mf.sh の halt) に組み込む停止コマンド

```
casstop /rESSRV01
killmfds.sh
```

cntl ファイルにて監視用のシェルを起動するように設定
設定を行なうノードで構成ファイルを利用できる共有ディスクを有効にする。
ES サーバを全て終了し、MFDS のみ起動させておく。

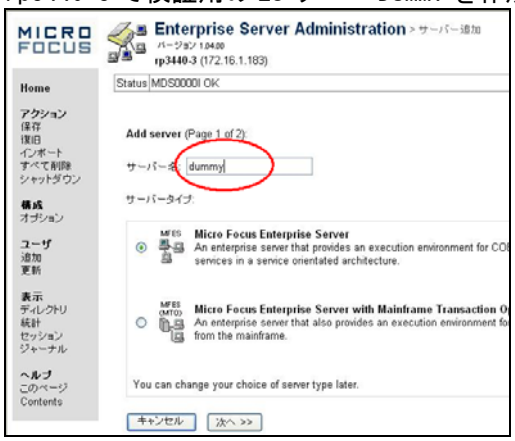
Enterprise Server Administration 画面のオプション(Option)のリポジトリロケーションを変更する。

構成リポジトリの格納ディレクトリはデフォルトでは、
　<Enterprise Server インストールディレクトリ>/etc/mfds/
に設定されている。
これを共有ファイルに設定する。本検証では /u01/mfds/ とした。
この操作により、/u01/mfds/ に新しい構成リポジトリが生成される。

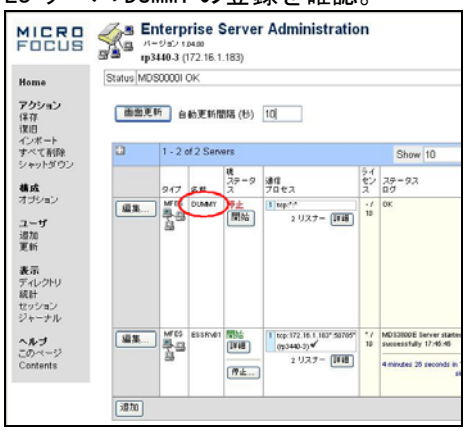
次に ES サーバ ESSRV01 にディプロイするサービスを共有ディスクに配置し、ディプロイパッケージ(.car ファイル)から再度ディプロイ作業を行なう。
本検証では/u01/deploy にディプロイパッケージを配置した

この作業を一つのノードで実施する。
これらの情報はフェイルオーバー時に引き継がれる。

＜検証＞
rp3440-3 で検証用の ES サーバ DUMMY を作成する。



ES サーバ DUMMY の登録を確認。

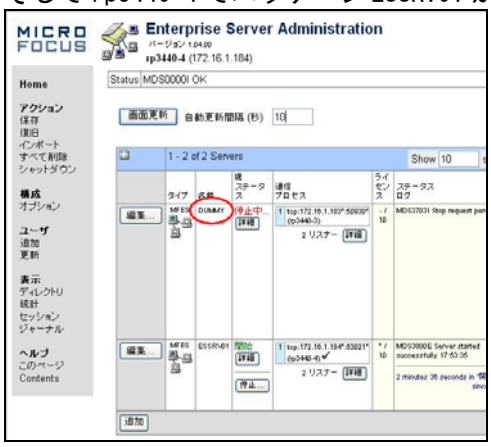


次に rp3440-3 の MFDS のプロセスを kill する。

- kill -9 <mfds32 の pid>

すると、ES 管理シェル mf.sh の **monitor**)が障害を検知しフェイルオーバーが開始する。

そして rp3440-4 でパッケージ ESSRV01 が開始される。



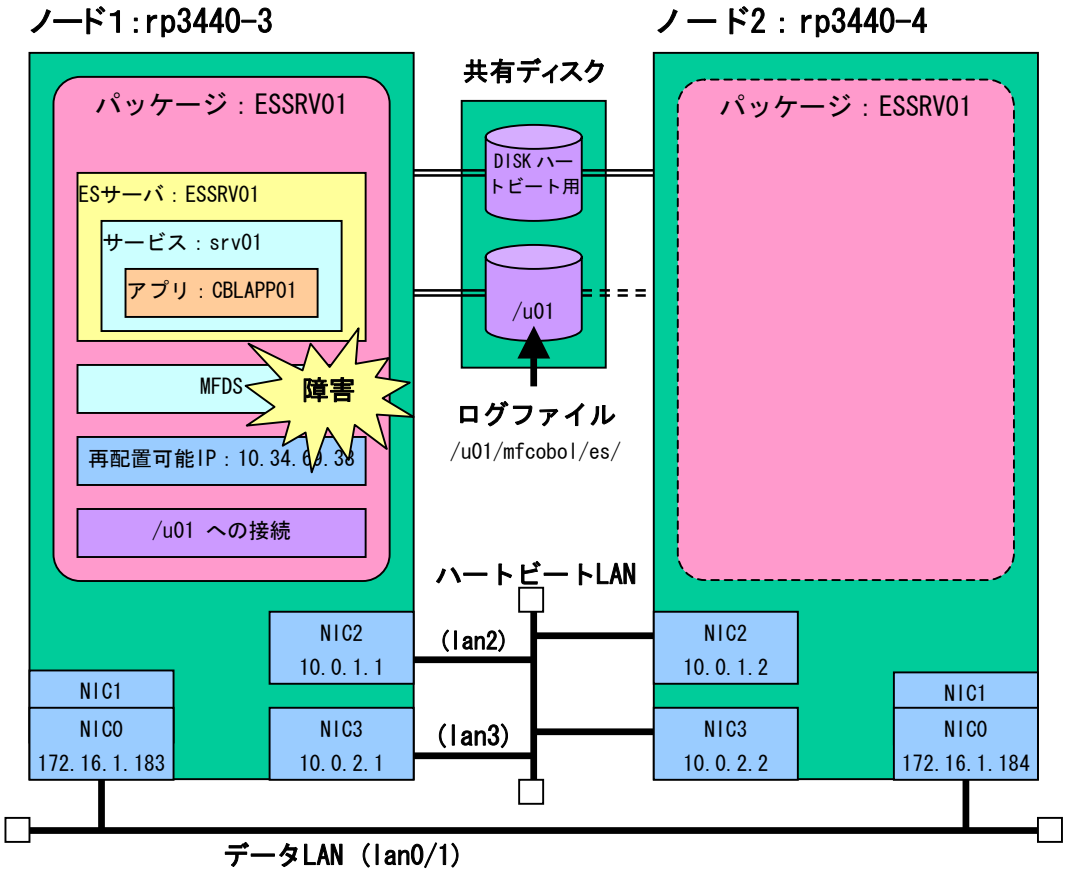
＜結果＞

共有ディスクに構成リポジトリを置く事で。2 つのノード間で構成リポジトリを共有して利用できた。またディプロイファイルも同時に共有することで、Enterprise Server の設定変更/ディプロイ作業を一元化できた。

パターン⑩ ログファイルの共有

＜目的＞
ES サーバが出力するログファイルの出力先を変更して共有化する。フェイルオーバー後に前ノードのログを参照できる事を確認する。

＜構成＞



＜設定のポイント＞
ES 管理シェル mf.sh の start) に組み込む起動コマンド

```
killmfds.sh
killallcas.sh ESSRV01
mfds &
casstart /rESSRV01
```

ES 管理シェル mf.sh の halt) に組み込む停止コマンド

```
casstop /rESSRV01
killmfds.sh
```

cntl ファイルにて監視用のシェルを起動するように設定

ログファイルは、デフォルトでは/var/mfcobol/es の下に各 ES サーバ毎に格納される。

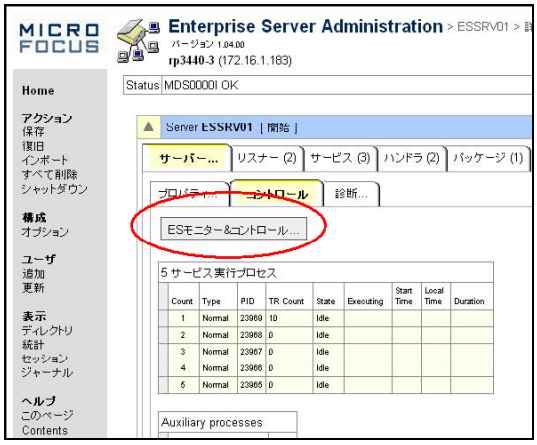
各ノードでログファイルの次の様に格納先を変更する。
共有ディスクを有効にして、MFDS、ES サーバを全て終了しておく。

シェル casperm を実行し対話で設定する。(実行は$COBDIR から。)
ここではログファイルは実行時システムファイルの一つとして見なして応答する。

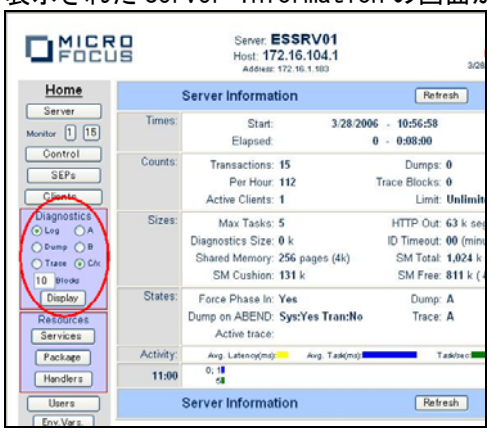
```
rp3440-3 # sh bin/casperm

続行する前に、Enterprise Server のシステム管理者用の英数字のユーザー ID を
手元に準備しているか確認してください。
詳細は、マニュアル「構成と管理」に記述されています。

終了するには、'q' を、続行するには、改行を押してください。:


次に続く質問に答えながら、グローバルトランザクションシステム
構成ファイル(cas.cfg)を構築します。- 詳細はマニュアル
「構成と管理」を参照してください。

エラーや警告メッセージを syslog デーモンから表示しますか
(y/n)?
y

Enterprise Server のシステム管理者用の英数字のユーザー ID を
入力してください。:root

Enterprise Server 実行時システムファイルをコピーするディレクトリを
絶対パス名で入力してください。環境変数は使用できません。
/var/mfcobol/es にコピーする場合は、RETURN キーを押してください。
/u01/mfcobol/es
ファイルパーミッションを設定中です。お待ちください...

Enterprise Server の構成を完了しました。

rp3440-3 #
```

＜検証＞
rp3440-3 でパッケージ ESSRV01 を立ち上げ、サービス要求などを数回行なう。

次に ES サーバ ESSRV01 が開始(Started)状態のまま、サーバ＞コントロールの画面のＥＳモニター＆コントロール ボタンをクリックする。



表示された Server Information の画面からログを参照する。



Log、C/x(現用ログ) の内容

```
ESSRV01
System Log: $TXRFDIR/console.log
060328 11155543                    CASCD0100I ES Daemon Initialized (Ver CAS 3.0.12) 11:15:55
060328 11155643                    CASCD0120I Server manager created for ES ESSRV01, process-id = 06141 11:15:
060328 11155646       6141 ESSRV01 CASSI0000I Server manager initialization started 11:15:56
060328 11155847       6141 ESSRV01 CASSI4005I Retrieving ES configuration from MFDS (127.0.0.1:86) 11:15:58
060328 11155953                    CASCD1020I JCP service process created for region ESSRV01, process-id = 061
060328 11155954                    CASCD0127I SEP 0001 created for ES ESSRV01, process-id = 06143 11:15:59
060328 11155956       6142 ESSRV01 CASJC0001I Journal control initialized 11:15:59
060328 11155959       6143 ESSRV01 CASSI1500I SEP initialization started 11:15:59
060328 11155961                    CASCD0127I SEP 0002 created for ES ESSRV01, process-id = 06144 11:15:59
060328 11155962       6143 ESSRV01 CASSI1600I SEP initialization completed successfully 11:15:59
060328 11155967       6144 ESSRV01 CASSI1500I SEP initialization started 11:15:59
060328 11155969                    CASCD0127I SEP 0003 created for ES ESSRV01, process-id = 06145 11:15:59
                                   .
                                   .
```

次に rp3440-3 の MFDS のプロセスを kill する。

- kill -9 <mfds32 の pid>

すると、ES 管理シェル mf.sh の **monitor**)が障害を検知しフェイルオーバーが開始する。
そして rp3440-4 でパッケージ ESSRV01 が開始される。

同様に Server Information の画面からログを参照する。
Log、B(1 世代前のログ) の内容

```
ESSRV01
System Log: $TXRFDIR/console.bak
060328 11155543                    CASCD0100I ES Daemon Initialized (Ver CAS 3.0.12) 11:15:55
060328 11155643                    CASCD0120I Server manager created for ES ESSRV01, process-id = 06141 11:15:
060328 11155646       6141 ESSRV01 CASSI0000I Server manager initialization started 11:15:56
060328 11155847       6141 ESSRV01 CASSI4005I Retrieving ES configuration from MFDS (127.0.0.1:86) 11:15:58
060328 11155953                    CASCD1020I JCP service process created for region ESSRV01, process-id = 061
060328 11155954                    CASCD0127I SEP 0001 created for ES ESSRV01, process-id = 06143 11:15:59
060328 11155956       6142 ESSRV01 CASJC0001I Journal control initialized 11:15:59
060328 11155959       6143 ESSRV01 CASSI1500I SEP initialization started 11:15:59
060328 11155961                    CASCD0127I SEP 0002 created for ES ESSRV01, process-id = 06144 11:15:59
060328 11155962       6143 ESSRV01 CASSI1600I SEP initialization completed successfully 11:15:59
060328 11155967       6144 ESSRV01 CASSI1500I SEP initialization started 11:15:59
060328 11155969                    CASCD0127I SEP 0003 created for ES ESSRV01, process-id = 06145 11:15:59
                                   .
                                   .
```

＜結果＞
共有ディスクにログファイル格納先を変更する事でフェイルオーバー後にログを参照することができた。但し、MFDS の立ち上げ時にログファイルは世代が変わるため、フェイルオーバー後は過去ログとしての参照となる。

まとめ

本検証では下記の 10 パターンについて動作を確認しました。

パターン	タイトル	テスト結果	確認事項
①	手動切り替え	○	SERVICEGUARD 標準機能で切り替え
②	データ LAN 両系(二重)障害	○	SERVICEGUARD 標準機能で切り替え
③	業務障害	○	MDUMP ユーティリティの出力を解析するシェルを自作して切り替え
④	OS のみで待機	○	MFDS 制御、ES サーバ制御をパッケージに含め切り替え
⑤	MFDS 起動で待機	○	ES サーバ制御をパッケージに含め切り替え
⑥	DS サーバ起動で待機	×	Enterprise Server のリスナーは定義する IP が立ち上げ時に実在しなければ立ち上がらないため、このパターンは実装不可
⑦	別の業務を提供しながら待機	○	パッケージを複数に分けて切り替え
⑧	ディプロイパッケージの共有	○	共有ディスクにディプロイパッケージを配置して切り替え
⑨	構成リポジトリの共有	○	ディプロイパッケージの共有と合わせ、共有ディスクに構成リポジトリを配置して切り替え
⑩	ログファイルの共有	○	共有ディスクにログファイルの出力先を指定して切り替え

異常検出のパターンでは、今回マイクロフォーカス提供の mdump ユーティリティを使用し、ステータスを監視する事で異常を検出することが出来ました。
本運用においては、ユーザプログラム又は関連ソフトウェアの監視なども含め、監視方法や監視対象を再検討する必要があります。

待機状態のパターンでは、待機系のノードでは ES サーバをあらかじめ立ち上げておく事ができない事が分かりました。
また MFDS は開始状態でも停止状態からもフェイルオーバーは可能でした。
但し、複数のパッケージで ES サーバを立ち上げるケース等では MFDS は ES サーバの起動前にあらかじめ起動しておく必要があるため、基本的には OS の起動時に MFDS を自動起動するように設定することをお奨めします。

Enterprise Server システムの情報の共有化のテストではディプロイパッケージ、構成リポジトリ、ログファイルの共有してフェイルオーバーすることを確認できました。

付録

サーバアプリケーションソース(cblapp01.cbl)

```
       FILE-CONTROL.
           SELECT APPFILE ASSIGN TO EXTERNAL APPFILE01
                  ORGANIZATION INDEXED RECORD KEY RKEY
                  ACCESS MODE RANDOM
                  FILE STATUS IS AP-STATUS.
       DATA                DIVISION.
       FILE                SECTION.
       FD  APPFILE.
       01  AF-REC.
           05  RKEY        PIC X(6).
           05  AF-COUNTER   PIC 9(10).
       WORKING-STORAGE     SECTION.
       01  WK-DATE         PIC X(6).
       01  WK-TIME         PIC X(8).
       01  AP-STATUS       pic x(02).
       LINKAGE             SECTION.
       01  LK-DATETIME     PIC X(20).
       01  LK-PGMNAME      PIC X(10).
       01  LK-HOSTNAME     PIC X(20).
       01  LK-COUNTER      PIC 9(10).
       PROCEDURE           DIVISION
             USING LK-DATETIME LK-PGMNAME LK-HOSTNAME LK-COUNTER.
       1.
           PERFORM    GET-DATETIME.
           MOVE       'CBLAPP01  '   TO LK-PGMNAME.
           CALL       'gethostname' USING LK-HOSTNAME BY VALUE 20.
           OPEN       I-O            APPFILE.
           MOVE       SPACE          TO RKEY.
           READ       APPFILE        INVALID KEY CONTINUE.
           MOVE       AF-COUNTER     TO LK-COUNTER.
           ADD        1              TO AF-COUNTER.
           REWRITE    AF-REC         INVALID KEY CONTINUE.
           CLOSE      APPFILE.
           EXIT       PROGRAM.

       GET-DATETIME.
           ACCEPT  WK-DATE  FROM DATE.
           ACCEPT  WK-TIME  FROM TIME.
           STRING  WK-DATE(1:2) '/' WK-DATE(3:2) '/' WK-DATE(5:2) ' '
                   WK-TIME(1:2) ':' WK-TIME(3:2) ':' WK-TIME(5:2) '.'
                   WK-TIME(7:2) DELIMITED BY SIZE INTO LK-DATETIME.
```

クライアントアプリケーションソース(clptest.java)

```
import com.microfocus.cobol.connector.spi.*;
import com.microfocus.cobol.connector.cci.*;
import com.microfocus.cobol.lang.*;

public class clptest
{
  public static void main(String[] args)
  {
          CobolNoTxManagedConnectionFactory  mcf;
          javax.resource.cci.ConnectionFactory cxf;
          javax.resource.cci.Connection connection=null;
          javax.resource.cci.Interaction interaction=null;

          try{
            mcf = new CobolNoTxManagedConnectionFactory();
            mcf.setServerHost("necp2");
            mcf.setServerPort("9003");
            mcf.setTrace(new Boolean(false));
            cxf = (javax.resource.cci.ConnectionFactory) mcf.createConnectionFactory();
            connection = cxf.getConnection();

            {
              javax.resource.cci.Interaction ix =
              connection.createInteraction();
              com.microfocus.cobol.connector.cci.CobolInteractionSpec
              iSpec = new
                    com.microfocus.cobol.connector.cci.CobolInteractionSpec();
              iSpec.setFunctionName("initialize");
              javax.resource.cci.RecordFactory rf = cxf.getRecordFactory();
              javax.resource.cci.IndexedRecord irec =
              rf.createIndexedRecord("beanArgs");
              irec.add(new Boolean(true));
              javax.resource.cci.Record orec = ix.execute(iSpec, irec);
              ix.close();
            }

            javax.resource.cci.Interaction ix = connection.createInteraction();
            CobolInteractionSpec iSpec = new CobolInteractionSpec();
            iSpec.setFunctionName("cblapp01S.CBLAPP01");
            iSpec.setArgument(0, java.lang.String.class, com.microfocus.cobol.RuntimeProperties.OUTPUT_ONLY);
            iSpec.setArgument(1, java.lang.String.class, com.microfocus.cobol.RuntimeProperties.OUTPUT_ONLY);
            iSpec.setArgument(2, java.lang.String.class, com.microfocus.cobol.RuntimeProperties.OUTPUT_ONLY);
            iSpec.setArgument(3, Long.class, com.microfocus.cobol.RuntimeProperties.OUTPUT_ONLY);
            javax.resource.cci.RecordFactory rf = cxf.getRecordFactory();
            javax.resource.cci.IndexedRecord iRec = rf.createIndexedRecord("CBLAPP01In");
            javax.resource.cci.IndexedRecord oRec = rf.createIndexedRecord("CBLAPP01Out");
            ix.execute(iSpec, iRec, oRec);
            ix.close();
            System.out.println(oRec.get(0) + " " + oRec.get(1) + " " + oRec.get(2) + " " + oRec.get(3));
          }
          catch(Exception e) {
                    System.err.println(e);
          }
          return;
  }

}
```