CLUSTERPRO
MC ProcessSaver 2.11 for Linux
OpenShift 連携ガイド

- 責任範囲
- 適用範囲
- 概要
- システム構成図
- 運用手順
- 設定手順
- 注意・制限事項

改版履歴

版数	改版	内容
1.0	2021.04	新規作成
2.0	2022.04	MC 2.7 に対応
3.0	2023.04	MC 2.8 に対応 適用範囲の更新
4.0	2024.04	MC 2.9 に対応
5.0	2025.04	MC 2.10 に対応
6.0	2026.04	MC 2.11に対応

はしがき

本書は、CLUSTERPRO MC ProcessSaver 2.11 for Linux (以後 ProcessSaver と記載します)を Red Hat OpenShift Container Platform (以後 OpenShift と記載します)上のコンテナーのヘルスチェックに使用する場合の設定手順について記載したものです。
本書では、コンテナーのメインプロセス、あるいはサブプロセスを監視対象プロセスとした設定を行います。

本書の設定によって OpenShift クラスター内のコンテナーの再起動および、コンテナーへのリクエスト停止が可能になります。

目次

1. 責任範囲

本書は、ProcessSaver を OpenShift アプリケーションのプローブを使用したコンテナーのヘルスチェックに用いるための注意点や設定例を参考情報として示すものであり、これらの動作保証を行うものではありません。

2. 適用範囲

ProcessSaver は、以下のコンテナプラットフォーム基盤に対応します。

- Red Hat OpenShift Container Platform 4.12 、4.14 、 4.15

本書の手順は、以下を使用して検証しています。
この他のバージョンのソフトウェアを使用した場合でも、いくつかの設定項目の読み替えで OpenShift との連携が可能です。

- Red Hat Enterprise Linux 8.5
- Red Hat OpenShift Container Platform 4.12
- CLUSTERPRO MC ProcessSaver 2.11 for Linux

参考ドキュメント

ProcessSaver を新規インストール・バージョンアップされる場合
→ "CLUSTERPRO MC ProcessSaver 2.11 for Linux リリースメモ" を参照してください。

ProcessSaver の基本機能を理解したい場合
→『CLUSTERPRO MC ProcessSaver 2.11 for Linux ユーザーズガイド』および
『CLUSTERPRO MC ProcessSaver for Linux 導入ガイド』を参照してください。

その他機能について知りたい場合
→ 関連マニュアルの内容をお読みいただき、目的のマニュアルを参照してください。

3. 概要

本書は、ProcessSaver を Red Hat OpenShift 上で使用する手順について説明したものです。

監視したいアプリケーション(プロセス)が起動するコンテナー内に ProcessSaver を導入することで、コンテナーの正常状態のチェックを行うことができます。

ProcessSaver は pcheck コマンドで、プロセスの死活監視およびプロセスのストール監視を行い、障害発生時にプロセスの再起動を行うことでコンテナーの復旧を行うことができます。
また、ProcessSaver ではこの監視状態を確認することが可能な padmin コマンドを提供しています。
この padmin コマンドを OpenShift のコンテナーの診断機能であるヘルスチェックに登録することで、コンテナーの正常性を定期的にチェックすることができます。
OpenShift が提供するヘルスチェック方式のうち、padmin コマンドを登録可能なものは以下となります。

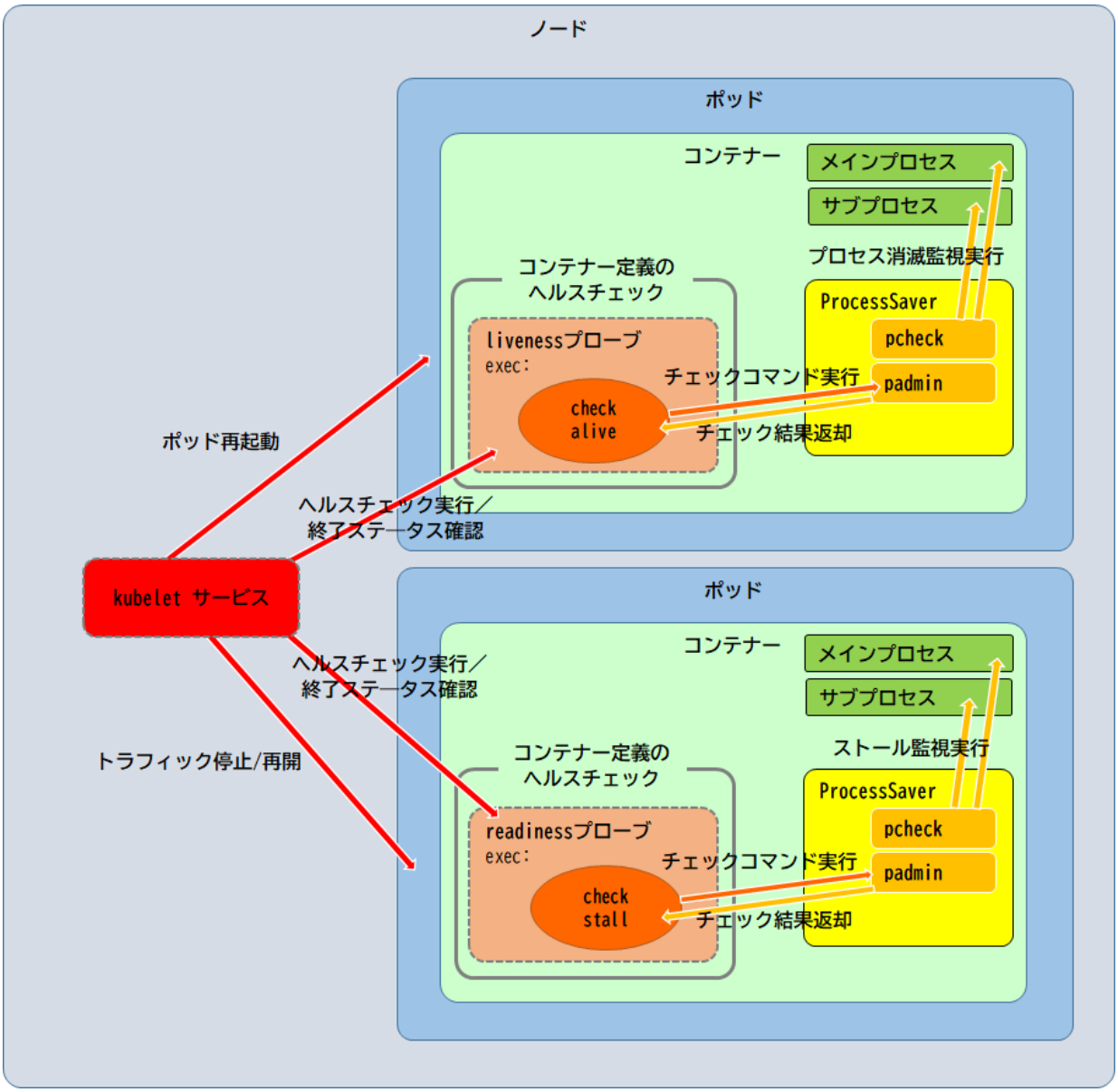
設定可否 (〇:設定可、×:設定不可)	使用するヘルスチェック方式
〇	livenessプローブ
〇	readinessプローブ
×	スタートアッププローブ

liveness プローブのチェック結果が異常であれば、OpenShift はコンテナーの再起動を行います。
readiness プローブのチェック結果が異常であれば、その結果が正常になるまで OpenShift は異常なコンテナープロセスに対してリクエストを止めるなどの対処を行います。
OpenShift で起動するプロセスに対して、pcheck コマンドを利用してプロセス監視を行い、その監視状態を定期的に確認する padmin コマンドと OpenShift の診断機能であるヘルスチェックを連携することで、OpenShift 上のコンテナーの正常性を確認することができます。
これによってさらに信頼性の高いサービスを提供することができます。

ProcessSaver を導入することで以下のような運用が可能となります。

① コンテナー内のサブプロセスの消滅およびストール発生時にコンテナーの再起動を行い、コンテナーを復旧することができます。
② コンテナー内のプロセスのストール発生時に、ストール状態が復旧するまで異常なコンテナーに対してリクエストを停止し、ストール状態から復旧した場合には、自動的にコンテナーを正常状態に復旧することができます。

4. システム構成図


ノード
ポッド
コンテナー
メインプロセス
サブプロセス
プロセス消滅監視実行
コンテナー定義の
ヘルスチェック
ProcessSaver
livenessプローブ
exec:
pcheck
チェックコマンド実行
padmin
check
alive
チェック結果返却
ポッド再起動
ヘルスチェック実行／
終了ステータス確認
kubelet サービス
ポッド
コンテナー
メインプロセス
サブプロセス
ヘルスチェック実行／
終了ステータス確認
ストール監視実行
トラフィック停止/再開
コンテナー定義の
ヘルスチェック
ProcessSaver
readinessプローブ
exec:
pcheck
チェックコマンド実行
padmin
check
stall
チェック結果返却

5. 運用手順

運用手順について説明します。

本書では、コンテナー上で起動するプロセスを以下のように定義します。

プロセス	PID	説明
メインプロセス	1	コンテナー上で常駐し停止するとコンテナーが終了するプロセス
サブプロセス	1以外	コンテナー上で常駐し停止してもコンテナーが終了しないプロセス

ProcessSaver を使用することで、OpenShift 上のコンテナーについてプロセス監視を行い、プロセス異常発生時にコンテナーを復旧する以下の 2 種類の運用が可能となります。

① コンテナー内のサブプロセスの消滅およびストール発生時にコンテナーの再起動を行い、コンテナーを復旧することが可能です。
ProcessSaver の死活監視機能、ストール監視機能と OpenShift のヘルスチェック機能である liveness プローブを連携することで実現します。

② コンテナー内のメインまたはサブプロセスのストールが発生した場合に、ストール状態が復旧するまで異常なコンテナーに対してリクエストを停止し、ストール状態から復旧した場合に、コンテナーを復旧することが可能です。
ProcessSaver のストール監視機能と OpenShift のヘルスチェック機能である readiness プローブを連携することで実現します。

5.1. livenessプローブを使用したヘルスチェック

OpenShift が提供するヘルスチェック機能(liveness プローブ)を使用することで、サブプロセスの消滅、ストール時にコンテナーの再起動を行い復旧することができます。

運用時の流れについては以下のとおりです。

1. ProcessSaver (pcheck コマンド)でコンテナー上のサブプロセスの死活監視/ストール監視
2. ProcessSaver(padmin コマンド)で監視ステータスを定期的にチェック
3. サブプロセスの消滅およびストール発生時にプロセス再起動を実行
4. OpenShift の liveness プローブで監視ステータスのチェック結果を確認
5. プロセスを再起動してもコンテナーの状態が復旧しない場合は、OpenShift がコンテナーを再起動
6. コンテナーを定常状態に復旧

5.2. readinessプローブを使用したヘルスチェック

OpenShift が提供するヘルスチェック機能(readiness プローブ)を使用することで、メイン/サブプロセスのストール時にストール状態が復旧するまで異常なコンテナーに対してのリクエストを停止することができます。
また、ストール状態から復旧した場合にコンテナーを定常状態に復旧することができます。

運用時の流れについては以下のとおりです。

1. ProcessSaver (pcheck コマンド)でコンテナー上のメイン/サブプロセスのストール監視
2. ProcessSaver(padmin コマンド)で監視ステータスを定期的にチェック
3. OpenShift の readiness プローブを使用して監視ステータスのチェック結果を確認
4. メイン/サブプロセスのストール発生時に OpenShift がコンテナーを準備(リクエスト不可)状態にする
5. プロセスのストール解消時には自動的にコンテナーを定常状態に復旧

5.3. livenessプローブ使用時のProcessSaverの監視設定

サブプロセスの死活監視またはストール監視を行う監視定義(pfile)を使用します。

pfile は以下の動作設定にする必要があります。

・サブプロセスの死活監視またはストール監視を設定

・プロセス消滅、ストール検出時にプロセスの再起動を行うための再起動スクリプトを設定

・リトライオーバー時に pcheck を終了する設定

監視定義(pfile)の詳細については、『CLUSTERPRO MC ProcessSaver 2.11 for Linux ユーザーズガイド』の「3.3. pfileファイルについて」および「4.6. ストール監視の導入手順」を参照してください。

5.4. readinessプローブ使用時のProcessSaverの監視設定

メインプロセスまたはサブプロセスのストール監視を行う監視定義(pfile)を使用します。

pfile は以下の動作設定にする必要があります。

・メインプロセスまたはサブプロセスのストール監視を設定

・再起動スクリプトは設定しない

・ストール検出時に対象プロセスを強制停止しない設定

・ストール検出時に継続してストール監視を実行する設定

監視定義(pfile)の詳細については、『CLUSTERPRO MC ProcessSaver 2.11 for Linux ユーザーズガイド』の「3.3. pfileファイルについて」および「4.6. ストール監視の導入手順」を参照してください。

5.5. livenessプローブ使用時のOpenShift 設定

死活監視状態チェックのため、運用管理コマンド(padmin) の check alive オプションを使用します。

OpenShift の liveness プローブは以下の動作設定にする必要があります。

・liveness プローブのコマンドに padmin コマンドの check alive オプションを設定

padmin コマンドの設定オプション、使用方法については、『CLUSTERPRO MC ProcessSaver 2.11 for Linux ユーザーズガイド』の「7. リファレンス」を参照してください。

5.6. readinessプローブ使用時のOpenShift 設定

ストール監視状態チェックのため、運用管理コマンド(padmin) の check stall オプションを使用します。

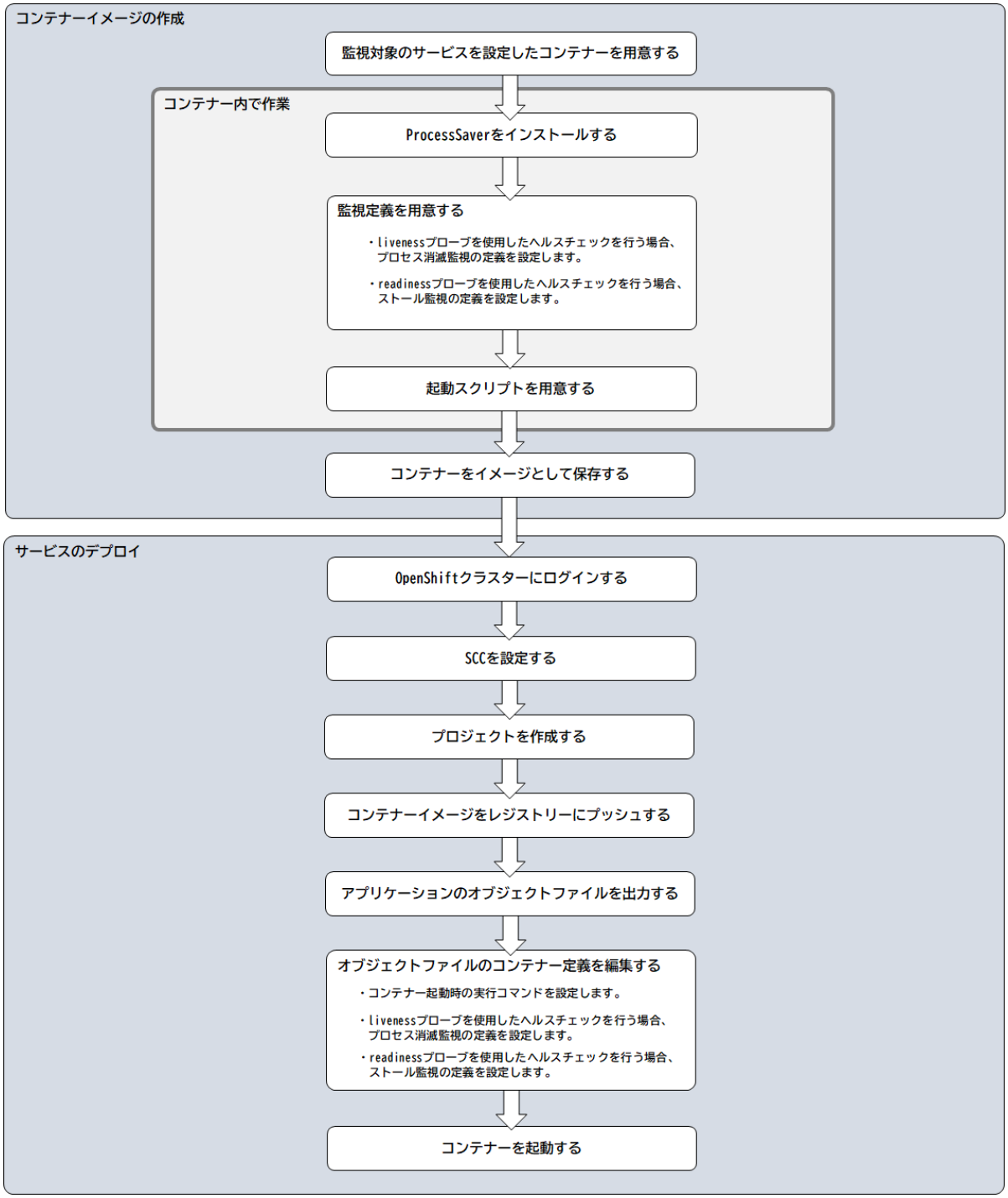
OpenShift の readiness プローブは以下の動作設定にする必要があります。

・readiness プローブのコマンドに padmin コマンドの check stall オプションを設定

padmin コマンドの設定オプション、使用方法については、『CLUSTERPRO MC ProcessSaver 2.11 for Linux ユーザーズガイド』の「7. リファレンス」を参照してください。

6. 設定手順

OpenShift クラスター内のコンテナーで動作するメインプロセス、またはサブプロセスの監視を ProcessSaver で行い、ProcessSaver のコマンドをコンテナーのヘルスチェックで利用する場合の手順を説明します。



おのおのの手順について以降のページで確認してください。

6.1. コンテナーイメージの作成

コンテナー内で対象の業務サービスとそのサービスを監視する ProcessSaver の動作設定を行い、イメージとして保存します。

6.1.1. 監視対象のサービスを設定したコンテナーを用意する

業務サービスを設定したコンテナー(以下「my-service」と呼ぶ)を用意します。

my-service の OS は、ProcessSaver が動作保証している OS に対応しています。
ProcessSaver が動作保証している OS の詳細については、『CLUSTERPRO MC ProcessSaver 2.11 for Linux リリースメモ』をご覧ください。

6.1.2. ProcessSaver をインストールする

my-serviceに ProcessSaver をインストールします。

本リリースのインストールについては、『CLUSTERPRO MC ProcessSaver 2.11 for Linux リリースメモ』をご覧ください。

6.1.3. 監視定義を用意する

ProcessSaver の監視定義(以下「pfile」と呼ぶ)を作成して my-service 内に配置します。

業務サービスの liveness チェックをする場合は、プロセスの消滅監視かストール監視の定義を作成してください。
業務サービスの readiness チェックをする場合は、プロセスのストール監視の定義を作成してください。

pfile の作成方法については、『CLUSTERPRO MC ProcessSaver 2.11 for Linux ユーザーズガイド』の「3.3. pfileファイルについて」とともに「4.6. ストール監視の導入手順」をご覧ください。

6.1.4. 起動スクリプトを用意する

my-service の起動時に、プロセス監視(プロセスの消滅監視およびストール監視を実行する PrecessSaver のコマンドのひとつで以下「pcheck」と呼ぶ)と業務サービスを順に起動するシェルスクリプトを作成します。

・/var/opt/HA/PS/conf/bin/ps_start.sh 記述例

```
#!/bin/sh
/opt/HA/PS/bin/pcheck -f pfile2 -w 10 &
/opt/HA/PS/bin/pcheck -f pfile1 -w 10 &

/home/username/my-app2.exe &
/home/username/my-app1.exe
```

[内容説明] pcheck を 2 個バックグラウンドで起動しています。
-w 10 は pcheck が起動してから 10 秒監視を待ち合わせます。
続いて業務サービスを2個起動します。
最後に起動する業務サービスには & がついていないので、my-service のメインプロセスとなります。
メインプロセスの終了は my-service の終了となります。

6.1.5. コンテナーをイメージとして保存する

my-service の設定が済んだところで my-service をイメージ(以下「 my-service-image 」と呼ぶ)として保存します。
OpenShift が my-service-image をダウンロードして同じ監視設定の my-service を再開できるようになります。

・コンテナーをイメージ化するコマンドの例

```
commit my-service my-service-image
```

[内容説明] コンテナー名 : my-service
イメージ名 : my-service-image

※podman コマンドは RHEL8.X の場合で示しています。

・イメージの一覧を確認するコマンドの例

```
$ podman images
```

6.2. サービスのデプロイ

作成した my-service-image からポッド定義オブジェクトファイルを作成し、ファイルの編集でコンテナー起動時に pcheck と業務サービスの起動スクリプトを実行する設定とともにコンテナーのヘルスチェックに ProcessServer の監視対象プロセスの状態チェック確認コマンド(以下「ヘルスチェックコマンド」と呼ぶ)を追加した後、 my-service として開始します。

6.2.1. OpenShift クラスターにログインする

管理者権限で OpenShift クラスターにログインします。

・管理者ログインの例

```
$ oc login -u kubeadmin -p APBEh-jjrVy-hLQZX-VI9Kg https://api.crc.testing:6443
Login successful.

You have access to 58 projects, the list has been suppressed. You can list all projects with ' projects'

Using project "default".
```

6.2.2. SCC (SECURITY CONTEXT CONSTRAINTS) を設定する

pcheck には root 権限が必要なのでデフォルトで anyuid の設定を行います。

・SCC設定の例

```
$ oc adm policy add-scc-to-user anyuid -z default
clusterrole.rbac.authorization.k8s.io/system:openshift:scc:anyuid added: "default"
```

新しくプロジェクトを作成した場合などは restricted ユーザーとなるので、これについてもSCCの設定を行います。

・SCC(restricted ユーザー)のRUNASUSER編集の例

```
$ oc edit scc restricted

【編集前】
     38 runAsUser:
     39   type: MustRunAsRange

【編集後】
     38 runAsUser:
     39   type: RunAsAny

securitycontextconstraints.security.openshift.io/restricted edited
```

・SCC編集後の確認の例

```
$ oc get scc
NAME              PRIV    CAPS         SELINUX     RUNASUSER         FSGROUP     SUPGROUP    PRIORITY
READONLYROOTFS   VOLUMES
<中略>
restricted        false   <no value>   MustRunAs   RunAsAny          MustRunAs   RunAsAny    <no value>
false            ["configMap","downwardAPI","emptyDir","persistentVolumeClaim","projected","secret"]
```

6.2.3. プロジェクトを作成する

新しいプロジェクトを開始します。

・新規プロジェクトを作成の例

```
$ oc new-project ps27
Now using project "default" on server "https://api.crc.testing:6443".

You can add applications to this project with the 'new-app' command. For example, try:

    oc new-app rails-postgresql-example

to build a new example application in Ruby. Or use kubectl to deploy a simple Kubernetes application:

    kubectl create deployment hello-node --image=k8s.gcr.io/serve_hostname

$ oc project
Using project "ps27" on server "https://api.crc.testing:6443".
```

6.2.4. コンテナーイメージをレジストリにプッシュする

前段「6.1.5. コンテナーをイメージとして保存する」の作業で生成した my-service-image をレポジトリに格納します。
この作業のために podman ログインします。

・podman ログインの例

```
$ sudo podman login -u kubeadmin -p $(oc whoami -t)
default-route-openshift-image-registry.apps-crc.testing --tls-verify=false
[sudo] developer のパスワード:
Login Succeeded!
```

レポジトリにアクセスするユーザーにロールを追加します。
イメージの書き出しやプッシュを実行するには (podman push コマンドを使用する場合など)、ユーザーに registry-editor ロールが必要です。

・ユーザーに registry-editor ロールを追加する例

```
$ oc policy add-role-to-user registry-editor developer
clusterrole.rbac.authorization.k8s.io/registry-editor added: “developer”
```

またイメージをプルするには (podman pull コマンドを使用する場合など)、ユーザーに registry-viewer ロールが必要です。

・ユーザーに registry-viewer ロールを追加する例

```
$ oc policy add-role-to-user registry-viewer developer
clusterrole.rbac.authorization.k8s.io/registry-viewer added: “developer”
```

イメージの一覧を表示します。
my-service-image にタグをつけます。

・イメージにタグ付けの例

```
$ sudo podman images
REPOSITORY                              TAG     IMAGE ID      CREATED      SIZE
docker.io/library/rhel7-ps27-liveness   latest  59e30202a49e  2 hours ago  246 MB
docker.io/library/rhel7-ps27-readiness  latest  5823dc0c0978  3 hours ago  250 MB
$
$
$ sudo podman tag docker.io/library/rhel7-ps27-liveness
default-route-openshift-image-registry.apps-crc.testing/ps27/rhel7-ps27-liveness:latest
$
$
$ sudo podman tag docker.io/library/rhel7-ps27-readiness
default-route-openshift-image-registry.apps-crc.testing/ps27/rhel7-ps27-readiness:latest
```

イメージの一覧からプロジェクトに必要なイメージにタグを付けました。
再度イメージの一覧を表示して確認します。

・イメージの一覧表示の例

```
$ sudo podman images
REPOSITORY                                                                  TAG     IMAGE ID
CREATED     SIZE
docker.io/library/rhel7-ps27-liveness                                         latest
59e30202a49e  2 hours ago  246 MB
default-route-openshift-image-registry.apps-crc.testing/ps27/rhel7-ps27-liveness   latest
59e30202a49e  2 hours ago  246 MB
docker.io/library/rhel7-ps27-readiness                                        latest
5823dc0c0978  3 hours ago  250 MB
default-route-openshift-image-registry.apps-crc.testing/ps27/rhel7-ps27-readiness  latest
5823dc0c0978  3 hours ago  250 MB
```

タグを付けた my-service-image をレジストリにプッシュします。

・イメージのプッシュの例

```
$ sudo podman push
default-route-openshift-image-registry.apps-crc.testing/ps27/rhel7-ps27-readiness
--tls-verify=false
Getting image source signatures
Copying blob 65dfb1d31a1e done
Writing manifest to image destination
Storing signatures
```

イメージストリームを取得し、プッシュされた my-service-image がリストされていることを確認します。

・イメージストリーム確認の例

```
$ oc get is
NAME                 IMAGE REPOSITORY
TAGS    UPDATED
rhel7-ps27-readiness
default-route-openshift-image-registry.apps-crc.testing/ps27/rhel7-ps27-readiness   latest   32
seconds ago
```

イメージストリーム内の my-service-image のタグのイメージルックアップを有効にします。

・イメージルックアップ有効化の例

```
$ oc set image-lookup
NAME                 LOCAL
rhel7-ps27-readiness  false
$
$ oc set image-lookup rhel7-ps27-readiness
imagestream.image.openshift.io/rhel7-ps27-readiness image lookup updated
$
$ oc set image-lookup
NAME                 LOCAL
rhel7-ps27-readiness  true
```

6.2.5. アプリケーションのオブジェクトファイルを出力する

イメージストリームから oc new-app で生成される OpenShift Container Platform オブジェクトをファイルに出力します。

・new-app アーティファクトをファイルに出力する例

```
$ oc new-app rhel7-ps27-readiness -o yaml > rhel7-ps27-pv-readiness.yaml
```

6.2.6. オブジェクトファイルのコンテナー定義を編集する

リダイレクト先のオブジェクトファイルを確認・編集し、業務サービスや pcheck の開始設定、デプロイ後のヘルスチェックの設定を追加していきます。

・OpenShift Container Platform オブジェクトファイルの確認の例

```
$ vi rhel7-ps27-pv-readiness.yaml
```

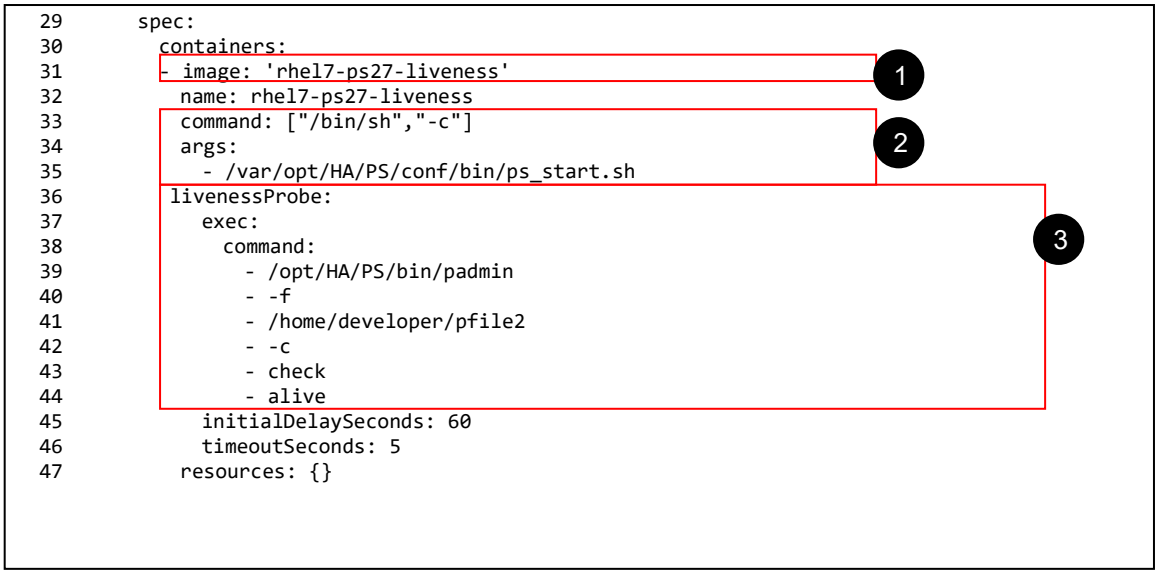
・編集前の状態

```
  1 warning: Cannot find git. Ensure that it is installed and in your path. Git is required to work
with git repositories.
  2 apiVersion: v1
  3 items:
  4 - apiVersion: apps/v1
  5   kind: Deployment
  6   metadata:
  7     annotations:
  8       image.openshift.io/triggers:
'[{"from":{"kind":"ImageStreamTag","name":"rhel7-ps27-readiness:latest","namespace":"ps27ft"},"
fieldPath":"spec.template.spec.containers[?(@.name==¥"rhel7-ps27-readiness¥")].image"}]'
  9       openshift.io/generated-by: OpenShiftNewApp
 10     creationTimestamp: null
 11     labels:
 12       app: rhel7-ps27-readiness
 13       app.kubernetes.io/component: rhel7-ps27-readiness
 14       app.kubernetes.io/instance: rhel7-ps27-readiness
 15     name: rhel7-ps27-readiness
 16   spec:
 17     replicas: 1
 18     selector:
 19       matchLabels:
 20         deployment: rhel7-ps27-readiness
 21     strategy: {}
 22     template:
 23       metadata:
 24         annotations:
 25           openshift.io/generated-by: OpenShiftNewApp
 26         creationTimestamp: null
 27         labels:
 28           deployment: rhel7-ps27-readiness
 29       spec:
 30         containers:
 31         - image: ' '
 32           name: rhel7-ps27-readiness
 33           resources: {}
 34   status: {}
 35 kind: List
 36 metadata: {}~
~
```

（1）livenessProbeの設定

オブジェクトファイルにコンテナー起動時のコマンドと起動後のヘルスチェック(liveness プローブ)の設定を追加します。

・編集後の状態(抜粋)



```
29      spec:
30        containers:
31        - image: 'rhel7-ps27-liveness'
32          name: rhel7-ps27-liveness
33          command: ["/bin/sh","-c"]
34          args:
35            - /var/opt/HA/PS/conf/bin/ps_start.sh
36          livenessProbe:
37            exec:
38              command:
39                - /opt/HA/PS/bin/padmin
40                - -f
41                - /home/developer/pfile2
42                - -c
43                - check
44                - alive
45            initialDelaySeconds: 60
46            timeoutSeconds: 5
47          resources: {}
```


コンテナーの定義を編集します。

① イメージ名を記載します。

② コンテナー起動時のコマンドを設定します。
この例では「args」部分に設定したパスのシェルスクリプトを実行します。
内容は「6.1.4. 起動スクリプトを用意する」で準備していたもので、pcheck を 2 個と、業務サービスを 2個起動します。

③ ヘルスチェックのプローブを設定します。
「liveness」プローブの実行方式「exec」の「command」部分で本リリースのヘルスチェックコマンドを指定しています。コマンドは pfile 「/home/developer/pfile2」の設定により my-service のサブプロセスの消滅監視を行っています。
この設定で my-service のサブプロセスが終了の際は「liveness」プローブのチェックを失敗させ、ポッドの再起動を行うことを企図しています。

（2）readinessProbeの設定

オブジェクトファイルにコンテナー起動時のコマンドと起動後のヘルスチェック(readiness プローブ)の設定を追加します。

・編集後の状態(抜粋)

```
29      spec:
30        containers:
31        - image: 'rhel7-ps27-readiness'                    ①
32          name: rhel7-ps27-readiness
33          command: ["/bin/sh","-c"]                        ②
34          args:
35            - /var/opt/HA/PS/conf/bin/ps_start.sh
36          readinessProbe:                                  ③
37            exec:
38              command:
39                - /opt/HA/PS/bin/padmin
40                - -f
41                - /home/developer/pfile2
42                - -c
43                - check
44                - stall
45            initialDelaySeconds: 60
46            timeoutSeconds: 5
47          resources: {}
```

コンテナーの定義を編集します。

① イメージ名を記載します。

② コンテナー起動時のコマンドを設定します。
この例では「args」部分に設定したパスのシェルスクリプトを実行します。
内容は「6.1.4. 起動スクリプトを用意する」で準備していたもので、pcheck を 2 個と、業務サービスを 2個起動します。

③ ヘルスチェックのプローブを設定します。
この場合は「readiness」プローブの実行方式「exec」の「command」部分で本リリースのヘルスチェックコマンドを指定しています。コマンドは pfile 「/home/developer/pfile2」の設定により my-service のサブプロセスのストール監視を行っています。
この設定で my-service のサブプロセスがストールの際は「readiness」プローブのチェックを失敗させ、 my-service へのリクエストを止めることを企図しています。

6.2.7. コンテナーを起動する

編集したオブジェクトファイルから oc create でアプリケーションの作成(OpenShift Container Platform オブジェクトを作成)を実行します。

・オブジェクトファイルからアプリケーション作成の例

```
$ oc create -f rhel7-ps27-readiness.yaml
deployment.apps/rhel7-ps27-readiness created
```

7. 注意・制限事項

・ProcessSaver を導入するコンテナーは root 権限を付与して起動する必要があります。
root 権限を付与する手順については、「6.2.2.　SCC (SECURITY CONTEXT CONSTRAINTS) を設定する」を参照してください。

CLUSTERPRO
MC ProcessSaver 2.11 for Linux
OpenShift 連携ガイド

2026 年 4 月 第 6 版
日本電気株式会社
東京都港区芝五丁目7番地1号
TEL (03) 3454-1111 (代表)

保護用紙