



Red Hat OpenShift Service on AWS 4

ノード

Red Hat OpenShift Service on AWS のノード

Red Hat OpenShift Service on AWS 4 ノード

Red Hat OpenShift Service on AWS のノード

法律上の通知

Copyright © 2024 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux[®] is the registered trademark of Linus Torvalds in the United States and other countries.

Java[®] is a registered trademark of Oracle and/or its affiliates.

XFS[®] is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL[®] is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js[®] is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack[®] Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

概要

本書では、クラスターのノード、Pod、コンテナを設定し管理する方法について説明します。また、Podのスケジューリングや配置の設定方法、ジョブやDaemonSetを使用してタスクを自動化する方法やクラスターを効率化するための他のタスクなどに関する情報も提供します。

目次

| | |
|---|------------|
| 第1章 ノードの概要 | 4 |
| 1.1. ノードについて | 4 |
| 1.2. POD について | 5 |
| 1.3. コンテナについて | 6 |
| 1.4. RED HAT OPENSIFT SERVICE ON AWS ノードの一般用語集 | 7 |
| 第2章 POD の使用 | 10 |
| 2.1. POD の使用 | 10 |
| 2.2. POD の表示 | 13 |
| 2.3. POD 用の RED HAT OPENSIFT SERVICE ON AWS の設定 | 18 |
| 2.4. シークレットを使用した機密データの POD への提供 | 24 |
| 2.5. 設定マップの作成および使用 | 49 |
| 2.6. POD スケジューリングの決定に POD の優先順位を含める | 66 |
| 2.7. ノードセクターの使用による特定ノードへの POD の配置 | 72 |
| 第3章 CUSTOM METRICS AUTOSCALER OPERATOR を使用した POD の自動スケーリング | 76 |
| 3.1. CUSTOM METRICS AUTOSCALER OPERATOR の概要 | 76 |
| 3.2. CUSTOM METRICS AUTOSCALER OPERATOR リリースノート | 78 |
| 3.3. カスタムメトリクスオートスケーラーのインストール | 84 |
| 3.4. カスタムメトリクスオートスケーラートリガーについて | 89 |
| 3.5. カスタムメトリクスオートスケーラートリガー認証について | 103 |
| 3.6. スケーリングされたオブジェクトのカスタムメトリクスオートスケーラーの一時停止 | 110 |
| 3.7. 監査ログの収集 | 112 |
| 3.8. デバッグデータの収集 | 117 |
| 3.9. OPERATOR メトリクスの表示 | 121 |
| 3.10. カスタムメトリクスオートスケーラーの追加方法について | 123 |
| 3.11. CUSTOM METRICS AUTOSCALER OPERATOR の削除 | 130 |
| 第4章 POD のノードへの配置の制御 (スケジューリング) | 133 |
| 4.1. スケジューラーによる POD 配置の制御 | 133 |
| 4.2. アフィニティールールと非アフィニティールールの使用による他の POD との相対での POD の配置 | 135 |
| 4.3. ノードのアフィニティールールを使用したノード上での POD 配置の制御 | 148 |
| 4.4. POD のオーバーコミットノードへの配置 | 158 |
| 4.5. ノードテイントを使用した POD 配置の制御 | 160 |
| 4.6. ノードセクターの使用による特定ノードへの POD の配置 | 181 |
| 4.7. POD トポロジー分散制約を使用した POD 配置の制御 | 191 |
| 第5章 ジョブと DEAMONSET の使用 | 196 |
| 5.1. デーモンセットによるノード上でのバックグラウンドタスクの自動的な実行 | 196 |
| 5.2. ジョブを使用した POD でのタスクの実行 | 200 |
| 第6章 ノードの使用 | 212 |
| 6.1. RED HAT OPENSIFT SERVICE ON AWS クラスター内のノードの閲覧とリスト表示 | 212 |
| 6.2. NODE TUNING OPERATOR の使用 | 220 |
| 6.3. ノードの修復、フェンシング、メンテナンス | 231 |
| 6.4. MACHINE CONFIG DAEMON メトリクス | 232 |
| 第7章 コンテナの使用 | 235 |
| 7.1. コンテナについて | 235 |
| 7.2. POD のデプロイ前の、INIT コンテナの使用によるタスクの実行 | 236 |
| 7.3. ボリュームの使用によるコンテナデータの永続化 | 241 |
| 7.4. PROJECTED ボリュームによるボリュームのマッピング | 254 |
| 7.5. コンテナによる API オブジェクト使用の許可 | 265 |

| | |
|--|------------|
| 7.6. RED HAT OPENSIFT SERVICE ON AWS コンテナ間でのファイルのコピー | 279 |
| 7.7. RED HAT OPENSIFT SERVICE ON AWS コンテナでのリモートコマンドの実行 | 282 |
| 7.8. コンテナ内のアプリケーションにアクセスするためのポート転送の使用 | 284 |
| 第8章 クラスターの操作 | 288 |
| 8.1. RED HAT OPENSIFT SERVICE ON AWS クラスター内のシステムイベント情報の表示 | 288 |
| 8.2. RED HAT OPENSIFT SERVICE ON AWS のノードが保持できる POD の数の見積り | 298 |
| 8.3. 制限範囲によるリソース消費の制限 | 305 |
| 8.4. コンテナメモリーとリスク要件を満たすためのクラスターメモリーの設定 | 317 |
| 8.5. オーバーコミットされたノード上に POD を配置するためのクラスターの設定 | 329 |

第1章 ノードの概要

1.1. ノードについて

ノードは、Kubernetes クラスター内の仮想マシンまたはベアメタルマシンです。ワーカーノードは、Podとしてグループ化されたアプリケーションコンテナをホストします。コントロールプレーンノードは、Kubernetes クラスターを制御するために必要なサービスを実行します。Red Hat OpenShift Service on AWS では、コントロールプレーンノードには、Red Hat OpenShift Service on AWS クラスターを管理するための Kubernetes サービス以上のものが含まれています。

クラスター内に安定した正常なノードを持つことは、ホストされたアプリケーションがスムーズに機能するための基本です。Red Hat OpenShift Service on AWS では、ノードを表す **Node** オブジェクトを介してノードにアクセス、管理、およびモニターできます。OpenShift CLI (**oc**) または Web コンソールを使用して、ノードで以下の操作を実行できます。

ノードの次のコンポーネントは、Pod の実行を維持し、Kubernetes ランタイム環境を提供するロールを果たします。

コンテナランタイム

コンテナランタイムはコンテナの実行を担当します。Kubernetes は、containerd、cri-o、rktlet、Docker などのいくつかのランタイムを提供します。

Kubelet

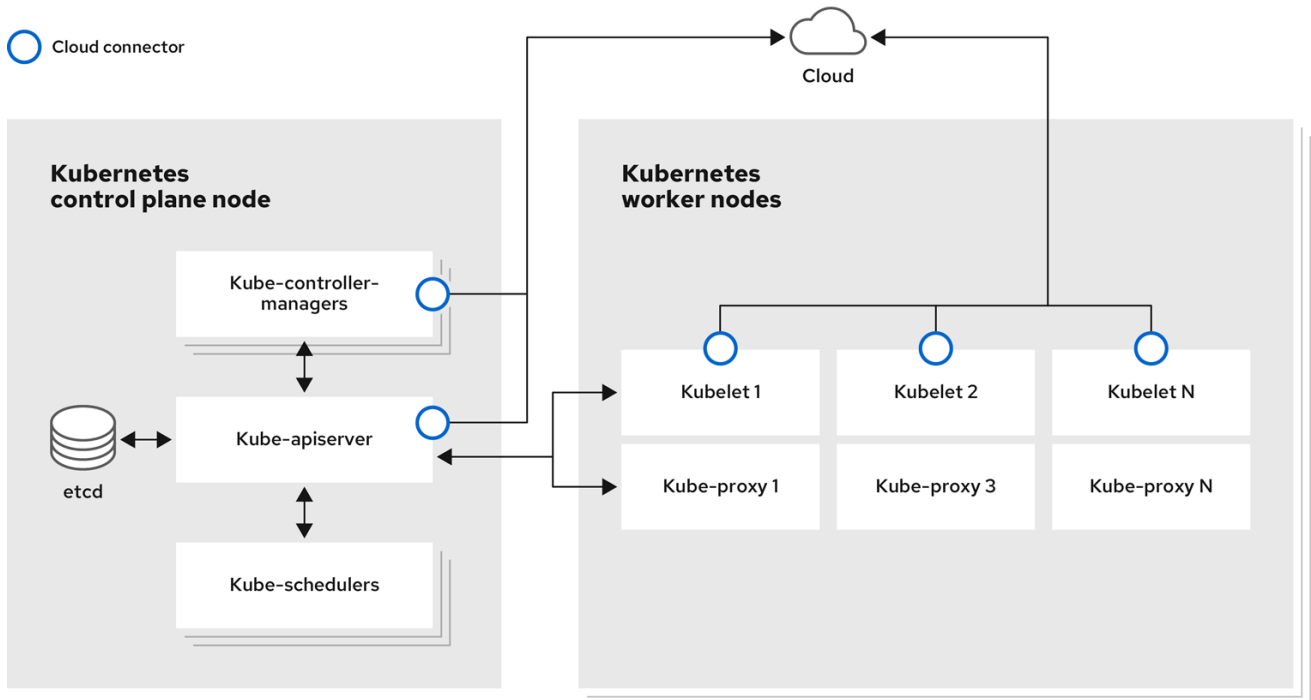
Kubelet はノード上で実行され、コンテナマニフェストを読み取ります。定義されたコンテナが開始され、実行されていることを確認します。kubelet プロセスは、作業の状態とノードサーバーを維持します。Kubelet は、ネットワークルールとポートフォワーディングを管理します。kubelet は、Kubernetes によってのみ作成されたコンテナを管理します。

Kube-proxy

Kube-proxy は、クラスター内のすべてのノードで実行され、Kubernetes リソース間のネットワークトラフィックを維持します。Kube プロキシは、ネットワーク環境が分離され、アクセス可能であることを保証します。

DNS

クラスター DNS は、Kubernetes サービスの DNS レコードを提供する DNS サーバーです。Kubernetes により開始したコンテナは、DNS 検索にこの DNS サーバーを自動的に含めます。



295_OpenShift_1222

読み取り操作

読み取り操作により、管理者または開発者は Red Hat OpenShift Service on AWS クラスタ内のノードに関する情報を取得できます。

- クラスタ内のすべてのノードを一覧表示します。
- メモリーと CPU の使用率、ヘルス、ステータス、経過時間など、ノードに関する情報を取得します。
- ノードで実行されている Pod を一覧表示します。

エンハンスメント操作

Red Hat OpenShift Service on AWS で可能なことは、ノードへのアクセスと管理だけではありません。管理者は、ノードで次のタスクを実行して、クラスタをより効率的でアプリケーションに適したものにし、開発者により良い環境を提供できます。

- **Node Tuning Operator** を使用して、ある程度のカーネルチューニングを必要とする高性能アプリケーションのノードレベルのチューニングを管理します。
- **デーモンセット** を使用して、ノードでバックグラウンドタスクを自動的に実行します。デーモンセットを作成して使用し、共有ストレージを作成したり、すべてのノードでロギング Pod を実行したり、すべてのノードに監視エージェントをデプロイしたりできます。

1.2. POD について

Pod は、ノードと一緒にデプロイされる 1 つ以上のコンテナです。クラスタ管理者は、Pod を定義し、スケジューリングの準備ができて正常なノードで実行するように割り当て、管理することができます。コンテナが実行されている限り、Pod は実行されます。Pod を定義して実行すると、Pod を変更することはできません。Pod を操作するときに行える操作は次のとおりです。

読み取り操作

管理者は、次のタスクを通じてプロジェクト内の Pod に関する情報を取得できます。

- レプリカと再起動の数、現在のステータス、経過時間などの情報を含む、プロジェクトに関連付けられている Pod を一覧表示 します。
- CPU、メモリー、ストレージ消費量などの Pod 使用状況の統計を表示 します。

管理操作

次のタスクのリストは、管理者が Red Hat OpenShift Service on AWS で Pod を管理する方法の概要を示しています。

- Red Hat OpenShift Service on AWS で利用可能な高度なスケジューリング機能を使用して、Pod のスケジューリングを制御します。
 - Pod アフィニティー、ノードアフィニティー、アンチ アフィニティーなどのノードから Pod へのバインディングルール。
 - ノードラベルとセレクター。
 - テイントおよび容認。
 - Pod トポロジー分散制約。
- Pod コントローラーと再起動ポリシーを使用して、再起動後の Pod の動作を設定します。
- Pod で送信トラフィックと受信トラフィックの両方を制限 します。
- Pod テンプレートを持つ任意のオブジェクトにボリュームを追加および削除します。ボリュームは、Pod 内のすべてのコンテナで使用できるマウントされたファイルシステムです。コンテナの保管はエフェメラルなものです。ボリュームを使用して、コンテナデータを永続化できます。

エンハンスメント操作

Red Hat OpenShift Service on AWS で利用可能なさまざまなツールと機能を使用して、Pod をより簡単かつ効率的に操作できます。次の操作では、これらのツールと機能を使用して Pod をより適切に管理します。

- Secrets: 一部のアプリケーションでは、パスワードやユーザー名などの機密情報が必要です。管理者は **Secret オブジェクト** を使用して、機密データを Pod に提供できます。

1.3. コンテナについて

コンテナは、Red Hat OpenShift Service on AWS アプリケーションの基本ユニットであり、依存関係、ライブラリー、およびバイナリーとともにパッケージ化されたアプリケーションコードで構成されます。コンテナは、複数の環境、および物理サーバー、仮想マシン (VM)、およびプライベートまたはパブリッククラウドなどの複数のデプロイメントターゲット間に一貫性をもたらします。

Linux コンテナテクノロジーは、実行中のプロセスを分離し、指定されたリソースのみへのアクセスを制限するための軽量メカニズムです。管理者は、Linux コンテナで次のようなさまざまなタスクを実行できます。

- コンテナーとの間でファイルをコピーします。
- コンテナーが API オブジェクトを使用できるようにします。
- コンテナーでリモートコマンドを実行します。
- コンテナー内のアプリケーションにアクセスするには、ポートフォワーディングを使用します。

Red Hat OpenShift Service on AWS は、**Init コンテナー** と呼ばれる特殊なコンテナーを提供します。Init コンテナーは、アプリケーションコンテナーの前に実行され、アプリケーションイメージに存在しないユーティリティまたはセットアップスクリプトを含めることができます。Pod の残りの部分がデプロイされる前に、Init コンテナーを使用してタスクを実行できます。

ノード、Pod、およびコンテナーで特定のタスクを実行する以外に、Red Hat OpenShift Service on AWS クラスター全体を操作して、クラスターの効率とアプリケーション Pod の高可用性を維持できます。

1.4. RED HAT OPENSIFT SERVICE ON AWS ノードの一般用語集

この用語集では、ノードのコンテンツで使用される一般的な用語を定義しています。

Container

これは、ソフトウェアとそのすべての依存関係を設定する軽量で実行可能なイメージです。コンテナーはオペレーティングシステムを仮想化するため、データセンターからパブリックまたはプライベートクラウド、さらには開発者のラップトップまで、どこでもコンテナーを実行できます。

デーモンセット

Pod のレプリカが Red Hat OpenShift Service on AWS クラスター内の対象となるノードで実行されるようにします。

egress

Pod からのネットワークのアウトバウンドトラフィックを介して外部とデータを共有するプロセス。

ガベージコレクション

終了したコンテナや実行中の Pod によって参照されていないイメージなどのクラスターリソースをクリーンアップするプロセス。

Ingress

Pod への着信トラフィック。

ジョブ

完了するまで実行されるプロセス。ジョブは 1 つ以上の Pod オブジェクトを作成し、指定された Pod が正常に完了するようにします。

ラベル

キーと値のペアであるラベルを使用して、Pod などのオブジェクトのサブセットを整理および選択できます。

ノード

Red Hat OpenShift Service on AWS クラスター内のワーカーマシンです。ノードは、仮想マシン (VM) または物理マシンのいずれかになります。

Node Tuning Operator

Node Tuning Operator を使用すると、TuneD デーモンを使用してノードレベルのチューニングを管理できます。これにより、カスタムチューニング仕様が、デーモンが認識する形式でクラスターで実行されるすべてのコンテナ化された TuneD デーモンに渡されます。デーモンは、ノードごとに 1 つずつ、クラスターのすべてのノードで実行されます。

Self Node Remediation Operator

Operator はクラスターノードで実行され、異常なノードを特定して再起動します。

Pod

Red Hat OpenShift Service on AWS クラスターで実行されている、ボリュームや IP アドレスなどの共有リソースを持つ 1 つ以上のコンテナ。Pod は、定義、デプロイ、および管理される最小のコンピュータ単位です。

容認

テイントが一致するノードまたはノードグループで Pod をスケジュールできる (必須ではない) ことを示します。容認を使用して、スケジューラーが一致するテイントを持つ Pod をスケジュール

できるようにすることができます。

テイント

キー、値、および **Effect** で設定されるコアオブジェクト。テイントと容認が連携して、**Pod** が無関係なノードでスケジュールされないようにします。

第2章 POD の使用

2.1. POD の使用

Pod は 1 つのホストにデプロイされる 1 つ以上のコンテナであり、定義され、デプロイされ、管理される最小のコンピュータ単位です。

2.1.1. Pod について

Pod はコンテナに対してマシンインスタンス (物理または仮想) とほぼ同じ機能を持ちます。各 Pod は独自の内部 IP アドレスで割り当てられるため、そのポートスペース全体を所有し、Pod 内のコンテナはそれらのローカルストレージおよびネットワークを共有できます。

Pod にはライフサイクルがあります。それらは定義された後にノードで実行されるために割り当てられ、コンテナが終了するまで実行されるか、その他の理由でコンテナが削除されるまで実行されます。ポリシーおよび終了コードによっては、Pod は終了後に削除されるか、コンテナのログへのアクセスを有効にするために保持される可能性があります。

Red Hat OpenShift Service on AWS は Pod をほぼイミュータブルなものとして扱います。Pod の実行中に Pod 定義を変更することはできません。Red Hat OpenShift Service on AWS は、既存の Pod を終了し、変更された設定、ベースイメージ、またはその両方を使用して Pod を再作成することによって変更を実装します。Pod は拡張可能なものとしても処理されますが、再作成時に状態を維持しません。Pod は拡張可能なものとして処理されますが、再作成時に状態を維持しません。そのため、通常 Pod はユーザーから直接管理されるのではなく、ハイレベルのコントローラーで管理される必要があります。



警告

レプリケーションコントローラーによって管理されないベア Pod はノードの中断時に再スケジュールされません。

2.1.2. Pod 設定の例

Red Hat OpenShift Service on AWS は、Kubernetes の Pod の概念を活用しています。Pod は、1 つのホスト上に一緒にデプロイされる 1 つ以上のコンテナであり、定義、デプロイ、管理できる最小のコンピュータ単位です。

以下は Pod の定義例です。これは数多くの Pod の機能を示していますが、それらのほとんどは他のトピックで説明されるため、ここではこれらについて簡単に説明します。

Pod オブジェクト定義 (YAML)

```
kind: Pod
apiVersion: v1
metadata:
  name: example
  labels:
    environment: production
    app: abc ①
spec:
  restartPolicy: Always ②
  securityContext: ③
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers: ④
    - name: abc
      args:
        - sleep
        - "1000000"
      volumeMounts: ⑤
        - name: cache-volume
          mountPath: /cache ⑥
      image: registry.access.redhat.com/ubi7/ubi-init:latest ⑦
      securityContext:
        allowPrivilegeEscalation: false
        runAsNonRoot: true
        capabilities:
          drop: ["ALL"]
      resources:
        limits:
          memory: "100Mi"
          cpu: "1"
        requests:
          memory: "100Mi"
          cpu: "1"
  volumes: ⑧
    - name: cache-volume
      emptyDir:
        sizeLimit: 500Mi
```

Pod には 1 つまたは複数のラベルでタグ付けすることができ、このラベルを使用すると、一度の操作で Pod グループの選択や管理が可能になります。これらのラベルは、キー/値形式で metadata ハッシュに保存されます。

2

Pod 再起動ポリシーと使用可能な値の Always、OnFailure、および Never です。デフォルト値は Always です。

3

Red Hat OpenShift Service on AWS は、コンテナを特権付きコンテナとして実行するか、選択したユーザーとして実行するかどうかを指定するセキュリティーコンテキストを定義します。デフォルトのコンテキストには多くの制限がありますが、管理者は必要に応じてこれを変更できます。

4

containers は、1 つ以上のコンテナ定義の配列を指定します。

5

コンテナは、コンテナ内に外部ストレージボリュームをマウントする場所を指定します。

6

Pod に提供するボリュームを指定します。ボリュームは指定されたパスにマウントされます。コンテナのルート (/) や、ホストとコンテナで同じパスにはマウントしないでください。これは、コンテナに十分な特権が付与されている場合、ホストシステムを破壊する可能性があります (例: ホストの /dev/pts ファイル)。ホストをマウントするには、/host を使用するのが安全です。

7

Pod 内の各コンテナは、独自のコンテナイメージからインスタンス化されます。

8

Pod は、コンテナで使用できるストレージボリュームを定義します。

ファイル数が多い永続ボリュームを Pod に割り当てる場合、それらの Pod は失敗するか、起動に時間がかかる場合があります。詳細は、[When using Persistent Volumes with high file counts in OpenShift, why do pods fail to start or take an excessive amount of time to achieve "Ready" state?](#) を参照してください。



注記

この Pod 定義には、Pod が作成され、ライフサイクルが開始した後に Red Hat OpenShift Service on AWS によって自動的に設定される属性が含まれません。[Kubernetes Pod ドキュメント](#)には、Pod の機能および目的についての詳細が記載されています。

2.1.3. 関連情報

- Pod とストレージの詳細については、[Understanding persistent storage](#) と [Understanding ephemeral storage](#) を参照してください。

2.2. POD の表示

管理者として、クラスターで Pod を表示し、それらの Pod および全体としてクラスターの正常性を判別することができます。

2.2.1. Pod について

Red Hat OpenShift Service on AWS は、Kubernetes の Pod の概念を活用しています。Pod は、1つのホスト上に一緒にデプロイされる1つ以上のコンテナであり、定義、デプロイ、管理できる最小のコンピュータ単位です。Pod はコンテナに対するマシンインスタンス (物理または仮想) とほぼ同等のものです。

特定のプロジェクトに関連付けられた Pod のリストを表示したり、Pod についての使用状況の統計を表示したりすることができます。

2.2.2. プロジェクトでの Pod の表示

レプリカの数、Pod の現在のステータス、再起動の数および年数を含む、現在のプロジェクトに関連付けられた Pod のリストを表示できます。

手順

プロジェクトで Pod を表示するには、以下を実行します。

1. プロジェクトに切り替えます。

```
$ oc project <project-name>
```

2.

以下のコマンドを実行します。

```
$ oc get pods
```

以下に例を示します。

```
$ oc get pods
```

出力例

```
NAME                READY STATUS  RESTARTS  AGE
console-698d866b78-bnshf 1/1  Running  2         165m
console-698d866b78-m87pm 1/1  Running  2         165m
```

-o wide フラグを追加して、Pod の IP アドレスと Pod があるノードを表示します。

```
$ oc get pods -o wide
```

出力例

```
NAME                READY STATUS  RESTARTS  AGE  IP             NODE
NOMINATED NODE
console-698d866b78-bnshf 1/1  Running  2         166m  10.128.0.24  ip-10-0-152-71.ec2.internal <none>
console-698d866b78-m87pm 1/1  Running  2         166m  10.129.0.23  ip-10-0-173-237.ec2.internal <none>
```

2.2.3. Pod の使用状況についての統計の表示

コンテナのランタイム環境を提供する、Pod についての使用状況の統計を表示できます。これら

の使用状況の統計には CPU、メモリー、およびストレージの消費量が含まれます。

前提条件

- 使用状況の統計を表示するには、cluster-reader 権限が必要です。
- 使用状況の統計を表示するには、メトリクスをインストールしている必要があります。

手順

使用状況の統計を表示するには、以下を実行します。

1. 以下のコマンドを実行します。

```
$ oc adm top pods
```

以下に例を示します。

```
$ oc adm top pods -n openshift-console
```

出力例

```
NAME                CPU(cores) MEMORY(bytes)
console-7f58c69899-q8c8k 0m          22Mi
console-7f58c69899-xhbgg 0m          25Mi
downloads-594fccc94-bcxk8 3m          18Mi
downloads-594fccc94-kv4p6 2m          15Mi
```

2. ラベルを持つ Pod の使用状況の統計を表示するには、以下のコマンドを実行します。

```
$ oc adm top pod --selector="
```

フィルターに使用するセレクター (ラベルクエリー) を選択する必要があります。=、==、および != をサポートします。

以下に例を示します。

```
$ oc adm top pod --selector='name=my-pod'
```

2.2.4. リソースログの表示

OpenShift CLI (oc) および Web コンソールでさまざまなリソースのログを表示できます。ログの末尾から読み取られるログ。

前提条件

- OpenShift CLI (oc) へのアクセスがある。

手順 (UI)

1. Red Hat OpenShift Service on AWS コンソールで **Workloads** → **Pods** に移動するか、調査するリソースから **Pod** に移動します。



注記

ビルドなどの一部のリソースには、直接クエリーする Pod がありません。このような場合は、リソースの **Details** ページで **Logs** リンクを特定できます。

2. ドロップダウンメニューからプロジェクトを選択します。
3. 調査する **Pod** の名前をクリックします。
4. **Logs** をクリックします。

手順 (CLI)

- 特定の Pod のログを表示します。

```
$ oc logs -f <pod_name> -c <container_name>
```

-

ここでは、以下のようになります。

- f

オプション: ログに書き込まれている内容に沿って出力することを指定します。

<pod_name>

Pod の名前を指定します。

<container_name>

オプション: コンテナの名前を指定します。Pod に複数のコンテナがある場合は、コンテナ名を指定する必要があります。

以下に例を示します。

```
$ oc logs ruby-58cd97df55-mww7r
```

```
$ oc logs -f ruby-57f7f4855b-znl92 -c ruby
```

ログファイルの内容が出力されます。

-

特定のリソースのログを表示します。

```
$ oc logs <object_type>/<resource_name> ①
```

①

リソースタイプおよび名前を指定します。

以下に例を示します。

```
$ oc logs deployment/ruby
```

ログファイルの内容が出力されます。

2.3. POD 用の RED HAT OPENSIFT SERVICE ON AWS の設定

管理者として、Pod に対して効率的なクラスターを作成し、維持することができます。

クラスターの効率性を維持することにより、1 回のみ実行するように設計された Pod をいつ再起動するか、Pod が利用できる帯域幅をいつ制限するか、中断時に Pod をどのように実行させ続けるかなど、Pod が終了するときの動作をツールとして使用して必要な数の Pod が常に実行されるようにし、開発者により良い環境を提供することができます。

2.3.1. 再起動後の Pod の動作方法の設定

Pod 再起動ポリシーは、Pod のコンテナの終了時に Red Hat OpenShift Service on AWS が応答する方法を決定します。このポリシーは Pod のすべてのコンテナに適用されます。

以下の値を使用できます。

- Always - Pod** で正常に終了したコンテナの再起動を継続的に試みます。指数関数的なバックオフ遅延 (10 秒、20 秒、40 秒) は 5 分に制限されています。デフォルトは Always です。
- OnFailure: Pod** で失敗したコンテナの継続的な再起動を、5 分を上限として指数関数のバックオフ遅延 (10 秒、20 秒、40 秒) で試行します。
- Never: Pod** で終了したコンテナまたは失敗したコンテナの再起動を試行しません。Pod はただちに失敗し、終了します。

いったんノードにバインドされた Pod は別のノードにはバインドされなくなります。これは、Pod がノードの失敗後も存続するにはコントローラーが必要であることを示しています。

| 条件 | コントローラーのタイプ | 再起動ポリシー |
|--------------------------------|-----------------|-----------------------------------|
| (バッチ計算など) 終了することが予想される Pod | ジョブ | OnFailure または Never |
| (Web サービスなど) 終了しないことが予想される Pod | レプリケーションコントローラー | Always |

| 条件 | コントローラーのタイプ | 再起動ポリシー |
|-------------------|-------------|---------|
| マシンごとに1回実行される Pod | デーモンセット | すべて |

Pod のコンテナが失敗し、再起動ポリシーが **OnFailure** に設定される場合、Pod はノード上に留まり、コンテナが再起動します。コンテナを再起動させない場合には、再起動ポリシーの **Never** を使用します。

Pod 全体に障害が発生した場合、Red Hat OpenShift Service on AWS は新しい Pod を起動します。開発者は、アプリケーションが新規 Pod で再起動される可能性に対応してはなりません。とくに、アプリケーションは、一時的なファイル、ロック、以前の実行で生じた未完成の出力などを処理する必要があります。

注記

Kubernetes アーキテクチャーでは、クラウドプロバイダーからの信頼性のあるエンドポイントが必要です。クラウドプロバイダーが停止している場合、kubelet により Red Hat OpenShift Service on AWS の再起動が阻止されます。

基礎となるクラウドプロバイダーのエンドポイントに信頼性がない場合は、クラウドプロバイダー統合を使用してクラスターをインストールしないでください。クラスターを、非クラウド環境で実行する場合のようにインストールします。インストール済みのクラスターで、クラウドプロバイダー統合をオンまたはオフに切り替えることは推奨されていません。

Red Hat OpenShift Service on AWS が失敗したコンテナに再起動ポリシーを使用する方法の詳細は、Kubernetes ドキュメントの [Example States](#) を参照してください。

2.3.2. Pod で利用可能な帯域幅の制限

QoS (Quality-of-Service) トラフィックシェーピングを Pod に適用し、その利用可能な帯域幅を効果的に制限することができます。(Pod からの) Egress トラフィックは、設定したレートを超えるパケットを単純にドロップするポリシーによって処理されます。(Pod への) Ingress トラフィックは、データを効果的に処理できるようシェーピングでパケットをキューに入れて処理されます。Pod に設定する制限は、他の Pod の帯域幅には影響を与えません。

手順

Pod の帯域幅を制限するには、以下を実行します。

1.

オブジェクト定義 JSON ファイルを作成し、`kubernetes.io/ingress-bandwidth` および `kubernetes.io/egress-bandwidth` アノテーションを使用してデータトラフィックの速度を指定します。たとえば、Pod の egress および ingress の両方の帯域幅を 10M/s に制限するには、以下を実行します。

制限が設定された Pod オブジェクト定義

```
{
  "kind": "Pod",
  "spec": {
    "containers": [
      {
        "image": "openshift/hello-openshift",
        "name": "hello-openshift"
      }
    ]
  },
  "apiVersion": "v1",
  "metadata": {
    "name": "iperf-slow",
    "annotations": {
      "kubernetes.io/ingress-bandwidth": "10M",
      "kubernetes.io/egress-bandwidth": "10M"
    }
  }
}
```

2.

オブジェクト定義を使用して Pod を作成します。

```
$ oc create -f <file_or_dir_path>
```

2.3.3. Pod の Disruption Budget (停止状態の予算) を使用して起動している Pod の数を指定する方法

Pod 中断バジェットでは、メンテナンスのためのノードのドレインなど、運用中の Pod に対する安全制約を指定できます。

PodDisruptionBudget は、同時に起動している必要のあるレプリカの最小数またはパーセンテージを指定する API オブジェクトです。これらをプロジェクトに設定することは、ノードのメンテナンス (クラスターのスケールダウンまたはクラスターのアップグレードなどの実行) 時に役立ち、この設定は (ノードの障害時ではなく) 自発的なエビクションの場合にのみ許可されます。

PodDisruptionBudget オブジェクトの設定は、次の主要な部分で構成されます。

- 一連の Pod に対するラベルのクエリ機能であるラベルセクター。
- 同時に利用可能にする必要のある Pod の最小数を指定する可用性レベル。
 - `minAvailable` は、中断時にも常に利用可能である必要のある Pod 数です。
 - `maxUnavailable` は、中断時に利用不可にできる Pod 数です。



注記

`Available` は、`Ready=True` の状態にある Pod 数を指します。`ready=True` は、要求に対応でき、一致するすべてのサービスの負荷分散プールに追加する必要がある Pod を指します。

`maxUnavailable` の 0% または 0 あるいは `minAvailable` の 100%、ないしはレプリカ数に等しい値は許可されますが、これによりノードがドレイン (解放) されないようにブロックされる可能性があります。



警告

Red Hat OpenShift Service on AWS のすべてのマシン設定プールにおける `maxUnavailable` のデフォルト設定は 1 です。この値を変更せず、一度に 1 つのコントロールプレーンノードを更新することを推奨します。コントロールプレーンプールのこの値を 3 に変更しないでください。

以下を実行して、Pod の Disruption Budget をすべてのプロジェクトで確認することができます。

```
$ oc get poddisruptionbudget --all-namespaces
```

出力例

| NAMESPACE | NAME | MIN AVAILABLE | MAX |
|-------------------------------------|-----------------------------------|---------------|-----|
| UNAVAILABLE | ALLOWED DISRUPTIONS | AGE | |
| openshift-apiserver | openshift-apiserver-pdb | N/A | 1 |
| 121m | | | 1 |
| openshift-cloud-controller-manager | aws-cloud-controller-manager | 1 | N/A |
| 1 | 125m | | |
| openshift-cloud-credential-operator | pod-identity-webhook | 1 | N/A |
| 1 | 117m | | |
| openshift-cluster-csi-drivers | aws-ebs-csi-driver-controller-pdb | N/A | 1 |
| 1 | 121m | | |
| openshift-cluster-storage-operator | csi-snapshot-controller-pdb | N/A | 1 |
| 1 | 122m | | |
| openshift-cluster-storage-operator | csi-snapshot-webhook-pdb | N/A | 1 |
| 1 | 122m | | |
| openshift-console | console | N/A | 1 |
| 116m | | | 1 |
| #... | | | |

PodDisruptionBudget は、最低でも minAvailable Pod がシステムで実行されている場合は正常であるとみなされます。この制限を超えるすべての Pod はエビクションの対象となります。



注記

Pod の優先順位およびプリエンプレションの設定に基づいて、優先順位の低い Pod は Pod の Disruption Budget の要件を無視して削除される可能性があります。

2.3.3.1. Pod の Disruption Budget を使用して起動している Pod 数の指定

同時に起動している必要のあるレプリカの最小数またはパーセンテージは、PodDisruptionBudget オブジェクトを使用して指定します。

手順

Pod の Disruption Budget を設定するには、以下を実行します。

1. YAML ファイルを以下のようなオブジェクト定義で作成します。

apiVersion: policy/v1 **1**

```
kind: PodDisruptionBudget
metadata:
  name: my-pdb
spec:
  minAvailable: 2
  selector:
    matchLabels:
      name: my-pod
```

①

PodDisruptionBudget は policy/v1 API グループの一部です。

②

同時に利用可能である必要のある Pod の最小数。これには、整数またはパーセンテージ (例: 20%) を指定する文字列を使用できます。

③

一連のリソースに対するラベルのクエリー。matchLabels と matchExpressions の結果は論理的に結合されます。プロジェクト内のすべての Pod を選択するには、このパラメーターを空白のままにします (例: selector {})

または、以下を実行します。

```
apiVersion: policy/v1
kind: PodDisruptionBudget
metadata:
  name: my-pdb
spec:
  maxUnavailable: 25%
  selector:
    matchLabels:
      name: my-pod
```

①

PodDisruptionBudget は policy/v1 API グループの一部です。

②

同時に利用不可にできる Pod の最大数。これには、整数またはパーセンテージ (例: 20%) を指定する文字列を使用できます。

③

一連のリソースに対するラベルのクエリー。matchLabels と matchExpressions の結果は論理的に結合されます。プロジェクト内のすべての Pod を選択するには、このパラメーターを空白のままにします (例: selector {})

2.

以下のコマンドを実行してオブジェクトをプロジェクトに追加します。

```
$ oc create -f </path/to/file> -n <project_name>
```

2.4. シークレットを使用した機密データの POD への提供

関連情報

アプリケーションによっては、パスワードやユーザー名など開発者に使用させない秘密情報が必要になります。

管理者としてシークレット オブジェクトを使用すると、この情報を平文で公開することなく提供することが可能です。

2.4.1. シークレットについて

Secret オブジェクトタイプは、パスワード、Red Hat OpenShift Service on AWS クライアント設定ファイル、プライベートソースリポジトリ認証情報などの機密情報を保持するメカニズムを提供します。シークレットは機密内容を Pod から切り離します。シークレットはボリュームプラグインを使用してコンテナにマウントすることも、システムが Pod の代わりにシークレットを使用して各種アクションを実行することもできます。

キーのプロパティには以下が含まれます。

- シークレットデータはその定義とは別に参照できます。
- シークレットデータのボリュームは一時ファイルストレージ機能 (tmpfs) でサポートされ、ノードで保存されることはありません。
- シークレットデータは namespace 内で共有できます。

YAML Secret オブジェクト定義

```
apiVersion: v1
kind: Secret
metadata:
  name: test-secret
  namespace: my-namespace
type: Opaque ①
data: ②
  username: <username> ③
  password: <password>
stringData: ④
  hostname: myapp.mydomain.com ⑤
```

①

シークレットにキー名および値の構造を示しています。

②

`data` フィールドのキーに使用可能な形式については、[Kubernetes identifiers glossary](#) の `DNS_SUBDOMAIN` 値のガイドラインに従う必要があります。

③

`data` マップのキーに関連付けられる値は `base64` でエンコーディングされている必要があります。

④

`stringData` マップのエントリーが `base64` に変換され、このエントリーは自動的に `data` マップに移動します。このフィールドは書き込み専用です。この値は `data` フィールドでのみ返されません。

⑤

`stringData` マップのキーに関連付けられた値は単純なテキスト文字列で設定されます。

シークレットに依存する Pod を作成する前に、シークレットを作成する必要があります。

シークレットの作成時に以下を実行します。

- シークレットデータでシークレットオブジェクトを作成します。
- Pod のサービスアカウントをシークレットの参照を許可するように更新します。
- シークレットを環境変数またはファイルとして使用する Pod を作成します (secret リソースを使用)。

2.4.1.1. シークレットの種類

`type` フィールドの値で、シークレットのキー名と値の構造を指定します。このタイプを使用して、シークレットオブジェクトにユーザー名とキーの配置を実行できます。検証の必要がない場合には、デフォルト設定の `opaque` タイプを使用してください。

以下のタイプから 1 つ指定して、サーバー側で最小限の検証をトリガーし、シークレットデータに固有のキー名が存在することを確認します。

- `kubernetes.io/service-account-token`。サービスアカウントトークンを使用します。
- `kubernetes.io/basic-auth`。Basic 認証で使用します。
- `kubernetes.io/ssh-auth`。SSH キー認証で使用します。
- `kubernetes.io/tls`。TLS 認証局で使用します。

検証が必要ない場合には `type: Opaque` と指定します。これは、シークレットがキー名または値の規則に準拠しないという意味です。`opaque` シークレットでは、任意の値を含む、体系化されていない `key:value` ペアも利用できます。



注記

`example.com/my-secret-type` などの他の任意のタイプを指定できます。これらのタイプはサーバー側では実行されませんが、シークレットの作成者がその種類のキー/値の要件に従う意図があることを示します。

シークレットのさまざまなタイプの例については、シークレットの使用に関連するコードのサンプルを参照してください。

2.4.1.2. シークレットデータキー

シークレットキーは DNS サブドメインになければなりません。

2.4.1.3. 自動生成されたシークレット

デフォルトでは、Red Hat OpenShift Service on AWS はサービスアカウントごとに次のシークレットを作成します。

- **dockercfg** イメージプルシークレット
- サービスアカウントトークンシークレット



注記

Red Hat OpenShift Service on AWS 4.11 より前では、サービスアカウントの作成時に 2 つ目のサービスアカウントトークンシークレットが生成されていました。このサービスアカウントトークンシークレットは、Kubernetes API へのアクセスに使用されていました。

Red Hat OpenShift Service on AWS 4.11 以降、この 2 つ目のサービスアカウントトークンシークレットは作成されなくなりました。これは、LegacyServiceAccountTokenNoAutoGeneration アップストリーム Kubernetes フィーチャーゲートが有効になっており、Kubernetes API にアクセスするためのシークレットベースのサービスアカウントトークンの自動生成が停止されているためです。

4 にアップグレードした後も、既存のサービスアカウントトークンシークレットは削除されず、引き続き機能します。

このサービスアカウントトークンシークレットと Docker 設定イメージプルシークレットは、OpenShift イメージレジストリーをクラスタのユーザー認証および認可システムに統合するために必要です。

ただし、ImageRegistry 機能を有効にしていない場合、または Cluster Image Registry Operator の設定で統合された OpenShift イメージレジストリーを無効にしている場合、これらのシークレットはサービスアカウントごとに生成されません。



警告

自分で使用する場合は、これらの自動生成されたシークレットに依存しないようにしてください。これらは、今後の Red Hat OpenShift Service on AWS のリリースで削除される可能性があります。

バインドされたサービスアカウントトークンを取得するために、予測されたボリュームでワークロードが自動的に挿入されます。ワークロードに追加のサービスアカウントトークンが必要な場合は、ワークロードマニフェストに追加の予測ボリュームを追加します。バインドされたサービスアカウントトークンは、次の理由により、サービスアカウントトークンのシークレットよりも安全です。

- バインドされたサービスアカウントトークンには有効期間が制限されています。
- バインドされたサービスアカウントトークンには対象ユーザーが含まれます。
- バインドされたサービスアカウントトークンは Pod またはシークレットにバインドでき、バインドされたオブジェクトが削除されるとバインドされたトークンは無効になります。

詳細は、ボリュームプロジェクションを使用したバインドされたサービスアカウントトークンの設定を参照してください。

読み取り可能な API オブジェクト内の有効期限のないトークンのセキュリティ露出が許容される場合は、サービスアカウントトークンシークレットを手動で作成してトークンを取得することもできます。詳細は、サービスアカウントトークンシークレットの作成を参照してください。

関連情報

- サービスアカウントトークンシークレットの作成については、[サービスアカウントトークンシークレットの作成](#)を参照してください。

2.4.2. シークレットの作成方法

管理者は、開発者がシークレットに依存する Pod を作成できるよう事前にシークレットを作成しておく必要があります。

シークレットの作成時に以下を実行します。

1. 秘密にしておきたいデータを含む秘密オブジェクトを作成します。各シークレットタイプに必要な特定のデータは、以下のセクションで非表示になります。

不透明なシークレットを作成する YAML オブジェクトの例

```
apiVersion: v1
kind: Secret
metadata:
  name: test-secret
type: Opaque ①
data: ②
  username: <username>
  password: <password>
stringData: ③
  hostname: myapp.mydomain.com
secret.properties: |
  property1=valueA
  property2=valueB
```

①

シークレットのタイプを指定します。

②

エンコードされた文字列およびデータを指定します。

③

デコードされた文字列およびデータを指定します。

`data` フィールドまたは `stringdata` フィールドの両方ではなく、いずれかを使用してください。

2.

Pod のサービスアカウントをシークレットを参照するように更新します。

シークレットを使用するサービスアカウントの YAML

```
apiVersion: v1
kind: ServiceAccount
...
secrets:
- name: test-secret
```

3.

シークレットを環境変数またはファイルとして使用する Pod を作成します (secret ポリユームを使用)。

シークレットデータと共にポリユームのファイルが設定された Pod の YAML

```
apiVersion: v1
kind: Pod
metadata:
  name: secret-example-pod
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
    - name: secret-test-container
      image: busybox
      command: [ "/bin/sh", "-c", "cat /etc/secret-volume/*" ]
      volumeMounts: ①
        - name: secret-volume
          mountPath: /etc/secret-volume ②
          readOnly: true ③
      securityContext:
        allowPrivilegeEscalation: false
        capabilities:
          drop: [ALL]
  volumes:
    - name: secret-volume
```

```
secret:
  secretName: test-secret ④
restartPolicy: Never
```

①

シークレットが必要な各コンテナに `volumeMounts` フィールドを追加します。

②

シークレットが表示される未使用のディレクトリー名を指定します。シークレットデータマップの各キーは `mountPath` の下にあるファイル名になります。

③

`true` に設定します。`true` の場合、ドライバーに読み取り専用ボリュームを提供するように指示します。

④

シークレットの名前を指定します。

シークレットデータと共に環境変数が設定された Pod の YAML

```
apiVersion: v1
kind: Pod
metadata:
  name: secret-example-pod
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
    - name: secret-test-container
      image: busybox
      command: [ "/bin/sh", "-c", "export" ]
      env:
        - name: TEST_SECRET_USERNAME_ENV_VAR
          valueFrom:
            secretKeyRef: ①
              name: test-secret
              key: username
      securityContext:
```

```
allowPrivilegeEscalation: false
capabilities:
  drop: [ALL]
restartPolicy: Never
```

1

シークレットキーを使用する環境変数を指定します。

シークレットデータと環境変数が設定されたビルド設定の YAML

```
apiVersion: build.openshift.io/v1
kind: BuildConfig
metadata:
  name: secret-example-bc
spec:
  strategy:
    sourceStrategy:
      env:
        - name: TEST_SECRET_USERNAME_ENV_VAR
          valueFrom:
            secretKeyRef: 1
              name: test-secret
              key: username
      from:
        kind: ImageStreamTag
        namespace: openshift
        name: 'cli:latest'
```

1

シークレットキーを使用する環境変数を指定します。

2.4.2.1. シークレットの作成に関する制限

シークレットを使用するには、Pod がシークレットを参照する必要があります。シークレットは、以下の 3 つの方法で Pod で使用されます。

- コンテナの環境変数を事前に設定するために使用される。
- 1つ以上のコンテナにマウントされるボリュームのファイルとして使用される。
- Pod のイメージをプルする際に kubelet によって使用される。

ボリュームタイプのシークレットは、ボリュームメカニズムを使用してデータをファイルとしてコンテナに書き込みます。イメージプルシークレットは、シークレットを namespace のすべての Pod に自動的に挿入するためにサービスアカウントを使用します。

テンプレートにシークレット定義が含まれる場合、テンプレートで指定のシークレットを使用できるようにするには、シークレットのボリュームソースを検証し、指定されるオブジェクト参照が Secret オブジェクトを実際に参照していることを確認する必要があります。そのため、シークレットはこれに依存する Pod の作成前に作成されている必要があります。最も効果的な方法として、サービスアカウントを使用してシークレットを自動的に挿入することができます。

シークレット API オブジェクトは namespace にあります。それらは同じ namespace の Pod によってのみ参照されます。

個々のシークレットは 1MB のサイズに制限されます。これにより、apiserver および kubelet メモリーを使い切るような大規模なシークレットの作成を防ぐことができます。ただし、小規模なシークレットであってもそれらを数多く作成するとメモリーの消費につながります。

2.4.2.2. 不透明なシークレットの作成

管理者は、不透明なシークレットを作成できます。これにより、任意の値を含むことができる非構造化 key:value のペアを格納できます。

手順

1. コントロールプレーンノードの YAML ファイルに Secret オブジェクトを作成します。

以下に例を示します。

```
apiVersion: v1
kind: Secret
metadata:
```

```
name: mysecret
type: Opaque ❶
data:
  username: <username>
  password: <password>
```

❶

不透明なシークレットを指定します。

2.

以下のコマンドを使用して **Secret** オブジェクトを作成します。

```
$ oc create -f <filename>.yaml
```

3.

Pod でシークレットを使用するには、以下を実行します。

a.

シークレットの作成方法についてセクションに示すように、Pod のサービスアカウントを更新してシークレットを参照します。

b.

シークレットの作成方法についてに示すように、シークレットを環境変数またはファイル (**secret** ボリュームを使用) として使用する Pod を作成します。

関連情報

●

Pod でのシークレットの使用の詳細については、[シークレットの作成方法について](#) を参照してください。

2.4.2.3. サービスアカウントトークンシークレットの作成

管理者は、サービスアカウントトークンシークレットを作成できます。これにより、API に対して認証する必要のあるアプリケーションにサービスアカウントトークンを配布できます。



注記

サービスアカウントトークンシークレットを使用する代わりに、TokenRequest API を使用してバインドされたサービスアカウントトークンを取得することを推奨します。TokenRequest API から取得したトークンは、有効期間が制限されており、他の API クライアントが読み取れないため、シークレットに保存されているトークンよりも安全です。

TokenRequest API を使用できず、読み取り可能な API オブジェクトで有効期限が切れていないトークンのセキュリティーエクスポージャーが許容できる場合にのみ、サービスアカウントトークンシークレットを作成する必要があります。

バインドされたサービスアカウントトークンの作成に関する詳細は、以下の追加リソースセクションを参照してください。

手順

1. コントロールプレーンノードの YAML ファイルに Secret オブジェクトを作成します。

secret オブジェクトの例:

```
apiVersion: v1
kind: Secret
metadata:
  name: secret-sa-sample
annotations:
  kubernetes.io/service-account.name: "sa-name" ①
type: kubernetes.io/service-account-token ②
```

①

既存のサービスアカウント名を指定します。ServiceAccount と Secret オブジェクトの両方を作成する場合は、ServiceAccount オブジェクトを最初に作成します。

②

サービスアカウントトークンシークレットを指定します。

2. 以下のコマンドを使用して **Secret** オブジェクトを作成します。

```
$ oc create -f <filename>.yaml
```

3. **Pod** でシークレットを使用するには、以下を実行します。
 - a. シークレットの作成方法についてセクションに示すように、**Pod** のサービスアカウントを更新してシークレットを参照します。
 - b. シークレットの作成方法についてに示すように、シークレットを環境変数またはファイル (**secret** ボリュームを使用) として使用する **Pod** を作成します。

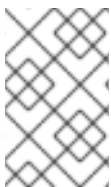
関連情報

- **Pod** でのシークレットの使用の詳細については、[シークレットの作成方法について](#) を参照してください。

2.4.2.4. Basic 認証シークレットの作成

管理者は **Basic** 認証シークレットを作成できます。これにより、**Basic** 認証に必要な認証情報を保存できます。このシークレットタイプを使用する場合は、**Secret** オブジェクトの **data** パラメーターには、**base64** 形式でエンコードされた以下のキーが含まれている必要があります。

- **username**: 認証用のユーザー名
- **password**: 認証のパスワードまたはトークン



注記

stringData パラメーターを使用して、クリアテキストコンテンツを使用できます。

手順

1. コントロールプレーンノードの **YAML** ファイルに **Secret** オブジェクトを作成します。

Secret オブジェクトの例

secret オブジェクトの例

```
apiVersion: v1
kind: Secret
metadata:
  name: secret-basic-auth
type: kubernetes.io/basic-auth 1
data:
stringData: 2
  username: admin
  password: <password>
```

1

Basic 認証のシークレットを指定します。

2

使用する Basic 認証値を指定します。

2.

以下のコマンドを使用して Secret オブジェクトを作成します。

```
$ oc create -f <filename>.yaml
```

3.

Pod でシークレットを使用するには、以下を実行します。

a.

シークレットの作成方法についてセクションに示すように、Pod のサービスアカウントを更新してシークレットを参照します。

b.

シークレットの作成方法についてに示すように、シークレットを環境変数またはファイル (secret ボリュームを使用) として使用する Pod を作成します。

関連情報

•

Pod でのシークレットの使用の詳細については、[シークレットの作成方法について](#) を参照してください。

2.4.2.5. SSH 認証シークレットの作成

管理者は、SSH 認証シークレットを作成できます。これにより、SSH 認証に使用されるデータを保存できます。このシークレットタイプを使用する場合、Secret オブジェクトの `data` パラメーターには、使用する SSH 認証情報が含まれている必要があります。

手順

1. コントロールプレーンノードの YAML ファイルに Secret オブジェクトを作成します。

secret オブジェクトの例:

```
apiVersion: v1
kind: Secret
metadata:
  name: secret-ssh-auth
type: kubernetes.io/ssh-auth ①
data:
  ssh-privatekey: | ②
    MIIEpQIBAAKCAQEAAulqb/Y ...
```

①

SSH 認証シークレットを指定します。

②

SSH のキー/値のペアを、使用する SSH 認証情報として指定します。

2. 以下のコマンドを使用して Secret オブジェクトを作成します。

```
$ oc create -f <filename>.yaml
```

3. Pod でシークレットを使用するには、以下を実行します。

- a. シークレットの作成方法についてセクションに示すように、Pod のサービスアカウントを更新してシークレットを参照します。

- b. シークレットの作成方法についてに示すように、シークレットを環境変数またはファイル (secret ボリュームを使用) として使用する Pod を作成します。

関連情報

- [シークレットの作成方法を理解する。](#)

2.4.2.6. Docker 設定シークレットの作成

管理者は Docker 設定シークレットを作成できます。これにより、コンテナイメージレジストリにアクセスするための認証情報を保存できます。

- [kubernetes.io/dockercfg](#).このシークレットタイプを使用してローカルの Docker 設定ファイルを保存します。secret オブジェクトの data パラメーターには、base64 形式でエンコードされた .dockercfg ファイルの内容が含まれている必要があります。
- [kubernetes.io/dockerconfigjson](#).このシークレットタイプを使用して、ローカルの Docker 設定 JSON ファイルを保存します。secret オブジェクトの data パラメーターには、base64 形式でエンコードされた .docker/config.json ファイルの内容が含まれている必要があります。

手順

1. コントロールプレーンノードの YAML ファイルに Secret オブジェクトを作成します。

Docker 設定の secret オブジェクトの例

```

apiVersion: v1
kind: Secret
metadata:
  name: secret-docker-cfg
  namespace: my-project
type: kubernetes.io/dockerconfig 1
data:
  .dockerconfig:bm5ubm5ubm5ubm5ubm5ubm5ubmdnZ2dnZ2dnZ2dnZ2dnZ2dnZ
2cgYXV0aCBrZXlzcG== 2

```

1

シークレットが Docker 設定ファイルを使用することを指定します。

2

base64 でエンコードされた Docker 設定ファイルの出力

Docker 設定の JSON secret オブジェクトの例

```
apiVersion: v1
kind: Secret
metadata:
  name: secret-docker-json
  namespace: my-project
type: kubernetes.io/dockerconfig 1
data:
  .dockerconfigjson:bm5ubm5ubm5ubm5ubm5ubm5ubmdnZ2dnZ2dnZ2dnZ2dnZ2
  dnZ2cgYXV0aCBrZXIzCg== 2
```

1

シークレットが Docker 設定の JSON ファイルを使用することを指定します。

2

base64 でエンコードされた Docker 設定 JSON ファイルの出力

2.

以下のコマンドを使用して Secret オブジェクトを作成します。

```
$ oc create -f <filename>.yaml
```

3.

Pod でシークレットを使用するには、以下を実行します。

- a. シークレットの作成方法についてセクションに示すように、Pod のサービスアカウントを更新してシークレットを参照します。
- b. シークレットの作成方法についてに示すように、シークレットを環境変数またはファイル (secret ボリュームを使用) として使用する Pod を作成します。

関連情報

- Pod でのシークレットの使用の詳細については、[シークレットの作成方法について](#) を参照してください。

2.4.2.7. Web コンソールを使用したシークレットの作成

Web コンソールを使用してシークレットを作成できます。

手順

1. **Workloads** → **Secrets** に移動します。
2. **Create** → **From YAML** をクリックします。
 - a. 仕様に合わせて **YAML** を手動で編集するか、ファイルを **YAML エディター** にドラッグアンドドロップします。以下に例を示します。

```
apiVersion: v1
kind: Secret
metadata:
  name: example
  namespace: <namespace>
type: Opaque ①
data:
  username: <base64 encoded username>
  password: <base64 encoded password>
stringData: ②
  hostname: myapp.mydomain.com
```

①

この例では、不透明なシークレットを指定します。ただし、サービスアカウントトークンシークレット、基本認証シークレット、SSH 認証シークレット、Docker 設定を使用するシークレットなど、他のシークレットタイプが表示される場合があります。

ます。

2

`stringData` マップのエントリーが `base64` に変換され、このエントリーは自動的に `data` マップに移動します。このフィールドは書き込み専用です。この値は `data` フィールドでのみ返されます。

3.

Create をクリックします。

4.

Add Secret to workload をクリックします。

a.

ドロップダウンメニューから、追加するワークロードを選択します。

b.

Save をクリックします。

2.4.3. シークレットの更新方法

シークレットの値を変更する場合、値 (すでに実行されている Pod で使用される値) は動的に変更されません。シークレットを変更するには、元の Pod を削除してから新規の Pod を作成する必要があります (同じ PodSpec を使用する場合があります)。

シークレットの更新は、新規コンテナイメージのデプロイメントと同じワークフローで実行されます。 `kubectl rolling-update` コマンドを使用できます。

シークレットの `resourceVersion` 値は参照時に指定されません。したがって、シークレットが Pod の起動と同じタイミングで更新される場合、Pod に使用されるシークレットのバージョンは定義されません。



注記

現時点で、Pod の作成時に使用されるシークレットオブジェクトのリソースバージョンを確認することはできません。コントローラーが古い `resourceVersion` を使用して Pod を再起動できるように、Pod がこの情報を報告できるようにすることが予定されています。それまでは既存シークレットのデータを更新せずに別の名前で新規のシークレットを作成します。

2.4.4. シークレットの作成および使用

管理者は、サービスアカウントトークンシークレットを作成できます。これにより、サービスアカウントトークンを API に対して認証する必要があるアプリケーションに配布できます。

手順

1. 以下のコマンドを実行して namespace にサービスアカウントを作成します。

```
$ oc create sa <service_account_name> -n <your_namespace>
```

2. 以下の YAML の例は `service-account-token-secret.yaml` という名前のファイルに保存します。この例には、サービスアカウントトークンの生成に使用可能な Secret オブジェクト設定が含まれています。

```
apiVersion: v1
kind: Secret
metadata:
  name: <secret_name> ①
  annotations:
    kubernetes.io/service-account.name: "sa-name" ②
type: kubernetes.io/service-account-token ③
```

①

<secret_name> は、サービストークンシークレットの名前に置き換えます。

②

既存のサービスアカウント名を指定します。ServiceAccount と Secret オブジェクトの両方を作成する場合は、ServiceAccount オブジェクトを最初に作成します。

③

サービスアカウントトークンシークレットタイプを指定します。

3. ファイルを適用してサービスアカウントトークンを生成します。

```
$ oc apply -f service-account-token-secret.yaml
```

4. 以下のコマンドを実行して、シークレットからサービスアカウントトークンを取得します。

```
$ oc get secret <sa_token_secret> -o jsonpath='{.data.token}' | base64 --decode ❶
```

出力例

```
ayJhbGciOiJSUzI1NiIsImtpZCI6IklOb2dtck1qZ3hCSWpoNnh5YnZhSE9QMkk3YnRZMV
ZocIFfQTZfRFp1YlUifQ.eyJpc3MiOiJrdWJlcm5ldGVzL3NlcnZpY2VhY2NvdW50liwia3Vi
ZXJuZXRlcy5pby9zZXJ2aWNIYWVudC9uYW1lc3BhY2UiOiJkZWZhdWx0liwia3ViZX
JuZXRlcy5pby9zZXJ2aWNIYWVudC9uYW1lc3BhY2UiOiJkZWZhdWx0liwia3ViZX
4tdHZrbnliLCJrdWJlcm5ldGVzLmlvL3NlcnZpY2VhY2NvdW50L3NlcnZpY2UtYWNjb3Vu
dC5uYW1lIjoiYnVpbGRlcilslmt1YmVybmV0ZXMuaW8vc2VydmliZWZjY291bnQvc2Vyd
mljZS1hY2NvdW50LnVpZCI6IjNmZGU2MGZmLTA1NGYtNDkyZi04YzhjLTNlZjE0NDk3M
mFmNyIsInN1Yil6InN5c3RibTpozXJ2aWNIYWVudDpkZWZhdWx0OmJ1aWxkZXl1f
Q.OmqFTDuMHC_IYvvEUrjr1x453hIEEHYcxS9VKSzmRkP1SiVZWPNPkTWIfNRp6bIUZD
3U6aN3N7dMMSN0el5hu36xPgpKTdvuckKLTcnelMx6cxOdAbrcw1mCmOCINscwjS1KO
1kzMtYnnq8rXHiMJELsNIhnRyyIXRTtNBsy4t64T3283s3SLsancyx0gy0ujx-
Ch3uKAKdZi5iT-I8jnnQ-ds5THDs2h65RJhgglQEmSxpHrLGZFmyHAQI-
_SjvmHZPXEc482x3SkaQHNLqpmrpJorNqh1M8ZHKzlujhZgVooMvJmWPXTb2vnnvi3DG
n2XI-hZxl1yD2yGH1RBpYUHA
```

❶

<sa_token_secret> は、サービストークンシークレットの名前に置き換えます。

5.

サービスアカウントトークンを使用して、クラスターの API で認証します。

```
$ curl -X GET <openshift_cluster_api> --header "Authorization: Bearer <token>" ❶
```

❷

❶

<openshift_cluster_api> は OpenShift クラスター API に置き換えます。

❷

<token> は、直前のコマンドで出力されるサービスアカウントトークンに置き換えます。

2.4.5. シークレットで署名証明書を使用する方法

サービスへの通信を保護するために、プロジェクト内のシークレットに追加できる署名済みのサービス証明書とキーのペアを生成するように Red Hat OpenShift Service on AWS を設定できます。

サービス提供証明書のシークレットは、追加設定なしの証明書を必要とする複雑なミドルウェアアプリケーションをサポートするように設計されています。これにはノードおよびマスターの管理者ツールで生成されるサーバー証明書と同じ設定が含まれます。

サービス提供証明書のシークレット用に設定されるサービス Pod 仕様

```
apiVersion: v1
kind: Service
metadata:
  name: registry
  annotations:
    service.beta.openshift.io/serving-cert-secret-name: registry-cert 1
# ...
```

1

証明書の名前を指定します。

他の Pod は Pod に自動的にマウントされる

`/var/run/secrets/kubernetes.io/serviceaccount/service-ca.crt` ファイルの CA バンドルを使用して、クラスターで作成される証明書 (内部 DNS 名の場合にのみ署名される) を信頼できます。

この機能の署名アルゴリズムは `x509.SHA256WithRSA` です。ローテーションを手動で実行するには、生成されたシークレットを削除します。新規の証明書が作成されます。

2.4.5.1. シークレットで使用する署名証明書の生成

署名されたサービス証明書/キーペアを Pod で使用するには、サービスを作成または編集して `service.beta.openshift.io/serving-cert-secret-name` アノテーションを追加した後に、シークレットを Pod に追加します。

手順

サービス提供証明書のシークレットを作成するには、以下を実行します。

1. サービスの Pod 仕様を編集します。
2. シークレットに使用する名前に `service.beta.openshift.io/serving-cert-secret-name` アノテーションを追加します。

```
kind: Service
apiVersion: v1
metadata:
  name: my-service
  annotations:
    service.beta.openshift.io/serving-cert-secret-name: my-cert 1
spec:
  selector:
    app: MyApp
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9376
```

証明書およびキーは PEM 形式であり、それぞれ `tls.crt` および `tls.key` に保存されます。

3. サービスを作成します。

```
$ oc create -f <file-name>.yaml
```

4. シークレットを表示して、作成されていることを確認します。
 - a. すべてのシークレットのリストを表示します。

```
$ oc get secrets
```

出力例

| NAME | TYPE | DATA | AGE |
|---------|-------------------|------|-----|
| my-cert | kubernetes.io/tls | 2 | 9m |

- b. シークレットの詳細を表示します。

```
$ oc describe secret my-cert
```

出力例

```
Name:      my-cert
Namespace: openshift-console
Labels:    <none>
Annotations: service.beta.openshift.io/expiry: 2023-03-08T23:22:40Z
            service.beta.openshift.io/originating-service-name: my-service
            service.beta.openshift.io/originating-service-uid: 640f0ec3-afc2-4380-bf31-
a8c784846a11
            service.beta.openshift.io/expiry: 2023-03-08T23:22:40Z

Type: kubernetes.io/tls

Data
====
tls.key: 1679 bytes
tls.crt: 2595 bytes
```

5. このシークレットを使用して Pod 仕様を編集します。

```
apiVersion: v1
kind: Pod
metadata:
  name: my-service-pod
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
  - name: mypod
    image: redis
    volumeMounts:
    - name: my-container
      mountPath: "/etc/my-path"
    securityContext:
      allowPrivilegeEscalation: false
```

```

capabilities:
  drop: [ALL]
volumes:
- name: my-volume
secret:
  secretName: my-cert
items:
- key: username
  path: my-group/my-username
  mode: 511

```

これが利用可能な場合、Pod が実行されます。この証明書は内部サービス DNS 名、`<service.name>.<service.namespace>.svc` に適しています。

証明書/キーのペアは有効期限に近づくと自動的に置換されます。シークレットの `service.beta.openshift.io/expiry` アノテーションで RFC3339 形式の有効期限の日付を確認します。



注記

ほとんどの場合、サービス DNS 名 `<service.name>.<service.namespace>.svc` は外部にルーティング可能ではありません。`<service.name>.<service.namespace>.svc` の主な使用方法として、クラスターまたはサービス間の通信用として、`re-encrypt` ルートで使用されます。

2.4.6. シークレットのトラブルシューティング

サービス証明書の生成は以下を出して失敗します (サービスの `service.beta.openshift.io/serving-cert-generation-error` アノテーションには以下が含まれます)。

```
secret/ssl-key references serviceUID 62ad25ca-d703-11e6-9d6f-0e9c0057b608, which does not match 77b6dd80-d716-11e6-9d6f-0e9c0057b60
```

証明書を生成したサービスがすでに存在しないか、サービスに異なる `serviceUID` があります。古いシークレットを削除し、サービスのアノテーション (`service.beta.openshift.io/serving-cert-generation-error`、`service.beta.openshift.io/serving-cert-generation-error-num`) をクリアして証明書の再生成を強制的に実行する必要があります。

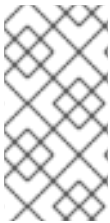
1. シークレットを削除します。

```
$ oc delete secret <secret_name>
```

2. アノテーションをクリアします。

```
$ oc annotate service <service_name> service.beta.openshift.io/serving-cert-generation-error-
```

```
$ oc annotate service <service_name> service.beta.openshift.io/serving-cert-generation-error-num-
```



注記

アノテーションを削除するコマンドでは、削除するアノテーション名の後に - を付けます。

2.5. 設定マップの作成および使用

以下のセクションでは、設定マップおよびそれらを作成し、使用方法を定義します。

2.5.1. 設定マップについて

数多くのアプリケーションには、設定ファイル、コマンドライン引数、および環境変数の組み合わせを使用した設定が必要です。Red Hat OpenShift Service on AWS では、これらの設定アーティファクトは、コンテナ化されたアプリケーションを移植可能な状態に保つためにイメージコンテンツから切り離されます。

ConfigMap オブジェクトは、コンテナを Red Hat OpenShift Service on AWS に依存させないようにする一方で、コンテナに設定データを挿入するメカニズムを提供します。設定マップは、個々のプロパティなどの粒度の細かい情報や、設定ファイル全体または JSON Blob などの粒度の荒い情報を保存するために使用できます。

ConfigMap オブジェクトは、Pod で使用したり、コントローラーなどのシステムコンポーネントの設定データを保存するために使用できる設定データのキーと値のペアを保持します。以下に例を示します。

ConfigMap オブジェクト定義

```
kind: ConfigMap
apiVersion: v1
metadata:
  creationTimestamp: 2016-02-18T19:14:38Z
```

```
name: example-config
namespace: my-namespace
data: ❶
example.property.1: hello
example.property.2: world
example.property.file: |-
  property.1=value-1
  property.2=value-2
  property.3=value-3
binaryData:
  bar: L3Jvb3QvMTAw ❷
```

❶ ❶

設定データが含まれます。

❷

バイナリー Java キーストアファイルなどの UTF8 以外のデータを含むファイルを参照します。Base 64 のファイルデータを入力します。



注記

イメージなどのバイナリーファイルから設定マップを作成する場合に、`binaryData` フィールドを使用できます。

設定データはさまざまな方法で Pod 内で使用できます。設定マップは以下を実行するために使用できます。

- コンテナへの環境変数値の設定
- コンテナのコマンドライン引数の設定
- ボリュームの設定ファイルの設定

ユーザーとシステムコンポーネントの両方が設定データを設定マップに保存できます。

設定マップはシークレットに似ていますが、機密情報を含まない文字列の使用をより効果的にサポートするように設計されています。

設定マップの制限

設定マップは、コンテンツを Pod で使用される前に作成する必要があります。

コントローラーは、設定データが不足していても、その状況を許容して作成できます。ケースごとに設定マップを使用して設定される個々のコンポーネントを参照してください。

ConfigMap オブジェクトはプロジェクト内にあります。

それらは同じプロジェクトの Pod によってのみ参照されます。

Kubelet は、API サーバーから取得する Pod の設定マップの使用のみをサポートします。

これには、CLI を使用して作成された Pod、またはレプリケーションコントローラーから間接的に作成された Pod が含まれます。これには、Red Hat OpenShift Service on AWS ノードの `--manifest-url` フラグ、`--config` フラグ、REST API を使用して作成された Pod は含まれません。これらは Pod を作成する一般的な方法ではないためです。

2.5.2. Red Hat OpenShift Service on AWS Web コンソールでの config map の作成

Red Hat OpenShift Service on AWS Web コンソールで config map を作成できます。

手順

- クラスター管理者として設定マップを作成するには、以下を実行します。
 1. **Administrator** パースペクティブで **Workloads** → **Config Maps** を選択します。
 2. ページの右上にある **Create Config Map** を選択します。

3. 設定マップの内容を入力します。

4. **Create** を選択します。

- 開発者として設定マップを作成するには、以下を実行します。

1. 開発者パースペクティブで、**Config Maps** を選択します。

2. ページの右上にある **Create Config Map** を選択します。

3. 設定マップの内容を入力します。

4. **Create** を選択します。

2.5.3. CLI を使用して設定マップを作成する

以下のコマンドを使用して、ディレクトリー、特定のファイルまたはリテラル値から設定マップを作成できます。

手順

- 設定マップの作成

```
$ oc create configmap <configmap_name> [options]
```

2.5.3.1. ディレクトリーからの設定マップの作成

`--from-file` フラグを使用すると、ディレクトリーから `config map` を作成できます。この方法では、ディレクトリー内の複数のファイルを使用して設定マップを作成できます。

ディレクトリー内の各ファイルは、`config map` にキーを設定するために使用されます。キーの名前はファイル名で、キーの値はファイルの内容です。

たとえば、次のコマンドは、`example-files` ディレクトリーの内容を使用して `config map` を作成します。

```
$ oc create configmap game-config --from-file=example-files/
```

`config map` 内のキーを表示します。

```
$ oc describe configmaps game-config
```

出力例

```
Name:      game-config
Namespace: default
Labels:    <none>
Annotations: <none>

Data

game.properties: 158 bytes
ui.properties:   83 bytes
```

マップにある2つのキーが、コマンドで指定されたディレクトリーのファイル名に基づいて作成されていることに気づかれることでしょう。これらのキーの内容は大きい可能性があるため、`oc description` の出力にはキーの名前とそのサイズのみが表示されます。

前提条件

- `config map` に追加するデータを含むファイルを含むディレクトリーが必要です。

次の手順では、サンプルファイル `game.properties` および `ui.properties` を使用します。

```
$ cat example-files/game.properties
```

出力例

```
enemies=aliens
lives=3
enemies.cheat=true
enemies.cheat.level=noGoodRotten
secret.code.passphrase=UDDLRBIBAS
secret.code.allowed=true
secret.code.lives=30
```

```
$ cat example-files/ui.properties
```

出力例

```
color.good=purple
color.bad=yellow
allow.textmode=true
how.nice.to.look=fairlyNice
```

手順

- 次のコマンドを入力して、このディレクトリー内の各ファイルの内容を保持する設定マップを作成します。

```
$ oc create configmap game-config \
  --from-file=example-files/
```

検証

- `-o` オプションを使用してオブジェクトの `oc get` コマンドを入力し、キーの値を表示します。

```
$ oc get configmaps game-config -o yaml
```

出力例

```
apiVersion: v1
data:
```

```

game.properties: |-
  enemies=aliens
  lives=3
  enemies.cheat=true
  enemies.cheat.level=noGoodRotten
  secret.code.passphrase=UUDDLRLRBABAS
  secret.code.allowed=true
  secret.code.lives=30
ui.properties: |
  color.good=purple
  color.bad=yellow
  allow.textmode=true
  how.nice.to.look=fairlyNice
kind: ConfigMap
metadata:
  creationTimestamp: 2016-02-18T18:34:05Z
  name: game-config
  namespace: default
  resourceVersion: "407"
  selflink: /api/v1/namespaces/default/configmaps/game-config
  uid: 30944725-d66e-11e5-8cd0-68f728db1985

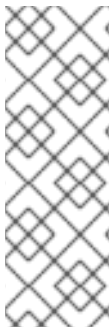
```

2.5.3.2. ファイルから設定マップを作成する

--from-file フラグを使用すると、ファイルから config map を作成できます。--from-file オプションを CLI に複数回渡すことができます。

key=value 式を --from-file オプションに渡すことで、ファイルからインポートされたコンテンツの config map に設定するキーを指定することもできます。以下に例を示します。

```
$ oc create configmap game-config-3 --from-file=game-special-key=example-
files/game.properties
```



注記

ファイルから設定マップを作成する場合、UTF8 以外のデータを破損することなく、UTF8 以外のデータを含むファイルをこの新規フィールドに配置できます。Red Hat OpenShift Service on AWS はバイナリーファイルを検出し、ファイルを MIME として透過的にエンコーディングします。サーバーでは、データを破損することなく MIME ペイロードがデコーディングされ、保存されます。

前提条件

- **config map に追加するデータを含むファイルを含むディレクトリーが必要です。**

次の手順では、サンプルファイル `game.properties` および `ui.properties` を使用します。

```
$ cat example-files/game.properties
```

出力例

```
enemies=aliens
lives=3
enemies.cheat=true
enemies.cheat.level=noGoodRotten
secret.code.passphrase=UDDLRBABS
secret.code.allowed=true
secret.code.lives=30
```

```
$ cat example-files/ui.properties
```

出力例

```
color.good=purple
color.bad=yellow
allow.textmode=true
how.nice.to.look=fairlyNice
```

手順

- **特定のファイルを指定して設定マップを作成します。**

```
$ oc create configmap game-config-2 \
  --from-file=example-files/game.properties \
  --from-file=example-files/ui.properties
```

- キーと値のペアを指定して、設定マップを作成します。

```
$ oc create configmap game-config-3 \
  --from-file=game-special-key=example-files/game.properties
```

検証

- `-o` オプションを使用してオブジェクトの `oc get` コマンドを入力し、ファイルからキーの値を表示します。

```
$ oc get configmaps game-config-2 -o yaml
```

出力例

```
apiVersion: v1
data:
  game.properties: |-
    enemies=aliens
    lives=3
    enemies.cheat=true
    enemies.cheat.level=noGoodRotten
    secret.code.passphrase=UDDLRRLRBABAS
    secret.code.allowed=true
    secret.code.lives=30
  ui.properties: |
    color.good=purple
    color.bad=yellow
    allow.textmode=true
    how.nice.to.look=fairlyNice
kind: ConfigMap
metadata:
  creationTimestamp: 2016-02-18T18:52:05Z
  name: game-config-2
  namespace: default
  resourceVersion: "516"
  selflink: /api/v1/namespaces/default/configmaps/game-config-2
  uid: b4952dc3-d670-11e5-8cd0-68f728db1985
```

- `-o` オプションを使用してオブジェクトの `oc get` コマンドを入力し、`key-value` (キー/値) ペアからキーの値を表示します。

```
$ oc get configmaps game-config-3 -o yaml
```

出力例

```

apiVersion: v1
data:
  game-special-key: |- 1
    enemies=aliens
    lives=3
    enemies.cheat=true
    enemies.cheat.level=noGoodRotten
    secret.code.passphrase=UUDDLRLRBABAS
    secret.code.allowed=true
    secret.code.lives=30
kind: ConfigMap
metadata:
  creationTimestamp: 2016-02-18T18:54:22Z
  name: game-config-3
  namespace: default
  resourceVersion: "530"
  selflink: /api/v1/namespaces/default/configmaps/game-config-3
  uid: 05f8da22-d671-11e5-8cd0-68f728db1985

```

1

これは、先の手順で設定したキーです。

2.5.3.3. リテラル値からの設定マップの作成

設定マップにリテラル値を指定することができます。

--from-literal オプションは、リテラル値をコマンドラインに直接指定できる key=value 構文を取ります。

手順

- リテラル値を指定して設定マップを作成します。

```

$ oc create configmap special-config \
  --from-literal=special.how=very \
  --from-literal=special.type=charm

```

検証

- `-o` オプションを使用してオブジェクトの `oc get` コマンドを入力し、キーの値を表示します。

```
$ oc get configmaps special-config -o yaml
```

出力例

```
apiVersion: v1
data:
  special.how: very
  special.type: charm
kind: ConfigMap
metadata:
  creationTimestamp: 2016-02-18T19:14:38Z
  name: special-config
  namespace: default
  resourceVersion: "651"
  selflink: /api/v1/namespaces/default/configmaps/special-config
  uid: dadce046-d673-11e5-8cd0-68f728db1985
```

2.5.4. ユースケース: Pod で設定マップを使用する

以下のセクションでは、Pod で ConfigMap オブジェクトを使用する際のいくつかのユースケースについて説明します。

2.5.4.1. 設定マップの使用によるコンテナでの環境変数の設定

`config map` を使用して、コンテナで個別の環境変数を設定するために使用したり、有効な環境変数名を生成するすべてのキーを使用してコンテナで環境変数を設定するために使用したりすることができます。

例として、以下の設定マップについて見てみましょう。

2つの環境変数を含む ConfigMap

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: special-config ①
  namespace: default ②
data:
  special.how: very ③
  special.type: charm ④
```

①

設定マップの名前。

②

設定マップが存在するプロジェクト。設定マップは同じプロジェクトの Pod によってのみ参照されます。

③ ④

挿入する環境変数。

1 つの環境変数を含む ConfigMap

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: env-config ①
  namespace: default
data:
  log_level: INFO ②
```

①

設定マップの名前。

②

手順

- configMapKeyRef セクションを使用して、Pod のこの ConfigMap のキーを使用できません。

特定の環境変数を挿入するように設定されている Pod 仕様のサンプル

```
apiVersion: v1
kind: Pod
metadata:
  name: dapi-test-pod
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
    - name: test-container
      image: gcr.io/google_containers/busybox
      command: [ "/bin/sh", "-c", "env" ]
      env: ①
        - name: SPECIAL_LEVEL_KEY ②
          valueFrom:
            configMapKeyRef:
              name: special-config ③
              key: special.how ④
        - name: SPECIAL_TYPE_KEY
          valueFrom:
            configMapKeyRef:
              name: special-config ⑤
              key: special.type ⑥
              optional: true ⑦
      envFrom: ⑧
        - configMapRef:
            name: env-config ⑨
      securityContext:
        allowPrivilegeEscalation: false
        capabilities:
          drop: [ALL]
      restartPolicy: Never
```

①

2

キーの値を挿入する Pod 環境変数の名前です。

3 5

特定の環境変数のプルに使用する ConfigMap の名前です。

4 6

ConfigMap からプルする環境変数です。

7

環境変数をオプションにします。オプションとして、Pod は指定された ConfigMap およびキーが存在しない場合でも起動します。

8

ConfigMap からすべての環境変数をプルするためのスタンザです。

9

すべての環境変数のプルに使用する ConfigMap の名前です。

この Pod が実行されると、Pod のログには以下の出力が含まれます。

```
SPECIAL_LEVEL_KEY=very
log_level=INFO
```



注記

SPECIAL_TYPE_KEY=charm は出力例にリスト表示されません。**optional: true** が設定されているためです。

2.5.4.2. 設定マップを使用したコンテナコマンドのコマンドライン引数の設定

config map を使用すると、Kubernetes 置換構文 **\$(VAR_NAME)** を使用してコンテナ内のコマンドまたは引数の値を設定できます。

例として、以下の設定マップについて見てみましょう。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: special-config
  namespace: default
data:
  special.how: very
  special.type: charm
```

手順

- コンテナ内のコマンドに値を挿入するには、環境変数として使用するキーを使用する必要があります。次に、`$(VAR_NAME)` 構文を使用してコンテナのコマンドでそれらを参照することができます。

特定の環境変数を挿入するように設定されている Pod 仕様のサンプル

```
apiVersion: v1
kind: Pod
metadata:
  name: dapi-test-pod
spec:
  securityContext:
    runAsNonRoot: true
  seccompProfile:
    type: RuntimeDefault
  containers:
    - name: test-container
      image: gcr.io/google_containers/busybox
      command: [ "/bin/sh", "-c", "echo $(SPECIAL_LEVEL_KEY)
$(SPECIAL_TYPE_KEY)" ] 1
      env:
        - name: SPECIAL_LEVEL_KEY
          valueFrom:
            configMapKeyRef:
              name: special-config
              key: special.how
        - name: SPECIAL_TYPE_KEY
          valueFrom:
            configMapKeyRef:
              name: special-config
              key: special.type
      securityContext:
        allowPrivilegeEscalation: false
```

```
capabilities:  
  drop: [ALL]  
restartPolicy: Never
```

1

環境変数として使用するキーを使用して、コンテナのコマンドに値を挿入します。

この Pod が実行されると、`test-container` コンテナで実行される `echo` コマンドの出力は以下ようになります。

```
very charm
```

2.5.4.3. 設定マップの使用によるボリュームへのコンテンツの挿入

設定マップを使用して、コンテンツをボリュームに挿入することができます。

ConfigMap カスタムリソース (CR) の例

```
apiVersion: v1  
kind: ConfigMap  
metadata:  
  name: special-config  
  namespace: default  
data:  
  special.how: very  
  special.type: charm
```

手順

設定マップを使用してコンテンツをボリュームに挿入するには、2つの異なるオプションを使用できます。

- 設定マップを使用してコンテンツをボリュームに挿入するための最も基本的な方法は、キーがファイル名であり、ファイルの内容がキーの値になっているファイルでボリュームを設定

定する方法です。

```
apiVersion: v1
kind: Pod
metadata:
  name: dapi-test-pod
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
    - name: test-container
      image: gcr.io/google_containers/busybox
      command: [ "/bin/sh", "-c", "cat", "/etc/config/special.how" ]
      volumeMounts:
        - name: config-volume
          mountPath: /etc/config
      securityContext:
        allowPrivilegeEscalation: false
        capabilities:
          drop: [ALL]
  volumes:
    - name: config-volume
      configMap:
        name: special-config ❶
  restartPolicy: Never
```

❶

キーを含むファイル。

この Pod が実行されると、cat コマンドの出力は以下のようになります。

```
very
```

- 設定マップキーが投影されるボリューム内のパスを制御することもできます。

```
apiVersion: v1
kind: Pod
metadata:
  name: dapi-test-pod
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
```

```

- name: test-container
  image: gcr.io/google_containers/busybox
  command: [ "/bin/sh", "-c", "cat", "/etc/config/path/to/special-key" ]
  volumeMounts:
  - name: config-volume
    mountPath: /etc/config
  securityContext:
    allowPrivilegeEscalation: false
    capabilities:
      drop: [ALL]
  volumes:
  - name: config-volume
    configMap:
      name: special-config
      items:
      - key: special.how
        path: path/to/special-key ①
  restartPolicy: Never

```

①

設定マップキーへのパス。

この Pod が実行されると、cat コマンドの出力は以下のようになります。

```

| very

```

2.6. POD スケジューリングの決定に POD の優先順位を含める

クラスターで Pod の優先順位およびプリエンプションを有効にできます。Pod の優先度は、他の Pod との比較した Pod の重要度を示し、その優先度に基づいて Pod をキューに入れます。Pod のプリエンプションは、クラスターが優先順位の低い Pod のエビクトまたはプリエンプションを実行することを可能にするため、適切なノードに利用可能な領域がない場合に優先順位のより高い Pod をスケジューリングできます。Pod の優先順位は Pod のスケジューリングの順序にも影響を与え、リソース不足の場合のノード上でのエビクションの順序に影響を与えます。

優先順位とプリエンプションを使用するには、Pod 仕様の優先順位クラスを参照して、その重みをスケジューリングに適用します。

2.6.1. Pod の優先順位について

Pod の優先順位およびプリエンプション機能を使用する場合、スケジューラーは優先順位に基づいて保留中の Pod を順序付け、保留中の Pod はスケジューリングのキューで優先順位のより低い他の保留中の Pod よりも前に置かれます。その結果、より優先順位の高い Pod は、スケジューリングの要件

を満たす場合に優先順位の低い Pod よりも早くスケジューラされる可能性があります。Pod をスケジューラできない場合、スケジューラは引き続き他の優先順位の低い Pod をスケジューラします。

2.6.1.1. Pod の優先順位クラス

Pod には優先順位クラスを割り当てることができます。これは、名前から優先順位の整数値へのマッピングを定義する namespace を使用していないオブジェクトです。値が高いと優先順位が高くなります。

優先順位およびプリエンプションは、1000000000 (10 億) 以下の 32 ビットの整数値を取ることができます。プリエンプションやエビクションを実行すべきでない Critical Pod 用に 10 億以上の数値を予約する必要があります。デフォルトで、Red Hat OpenShift Service on AWS には予約済みの優先順位クラスが 2 つあります。これらのクラスは、重要なシステム Pod のスケジューリングを保証するために使用されます。

```
$ oc get priorityclasses
```

出力例

| NAME | VALUE | GLOBAL-DEFAULT | AGE |
|-------------------------|------------|----------------|-------|
| system-node-critical | 2000001000 | false | 72m |
| system-cluster-critical | 2000000000 | false | 72m |
| openshift-user-critical | 1000000000 | false | 3d13h |
| cluster-logging | 1000000 | false | 29s |

- **system-node-critical:** この優先順位クラスには 2000001000 の値があり、ノードからエビクションすべきでないすべての Pod に使用されます。この優先順位クラスを持つ Pod の例として、sdn-ovs、sdn などがあります。数多くの重要なコンポーネントには、デフォルトで system-node-critical の優先順位クラスが含まれます。以下は例になります。
 - master-api
 - master-controller
 - master-etcd

- **sdn**
- **sdn-ovs**
- **sync**
- **system-cluster-critical:** この優先順位クラスには 2000000000 (20 億) の値があり、クラスターに重要な Pod に使用されます。この優先順位クラスの Pod は特定の状況でノードからエビクトされる可能性があります。たとえば、**system-node-critical** 優先順位クラスで設定される Pod が優先される可能性があります。この場合でも、この優先順位クラスではスケジューリングが保証されます。この優先順位クラスを持つ可能性のある Pod の例として、**fluentd**、**descheduler** などのアドオンコンポーネントなどがあります。数多くの重要なコンポーネントには、デフォルトで **system-cluster-critical** 優先順位クラスが含まれます。以下はその一例です。
 - **fluentd**
 - **metrics-server**
 - **descheduler**
- **openshift-user-critical:** `priorityClassName` フィールドを、リソース消費をバインドできず、予測可能なリソース消費動作がない重要な Pod で使用できます。**openshift-monitoring** および **openshift-user-workload-monitoring namespace** 下にある **Prometheus Pod** は、**openshift-user-critical priorityClassName** を使用します。モニタリングのワークロードは **system-critical** を最初の `priorityClass` として使用しますが、これにより、モニタリング時にメモリーが過剰に使用され、ノードがエビクトできない問題が発生します。その結果、モニタリングの優先順位が下がり、スケジューラーに柔軟性が与えられ、重要なノードの動作を維持するために重いワークロード発生します。
- **cluster-logging:** この優先順位は、**Fluentd Pod** が他のアプリケーションより優先してノードにスケジュールされるようにするために **Fluentd** で使用されます。

2.6.1.2. Pod の優先順位名

1 つ以上の優先順位クラスを準備した後に、Pod 仕様に優先順位クラス名を指定する Pod を作成できます。優先順位の受付コントローラーは、優先順位クラス名フィールドを使用して優先順位の整数値

を設定します。名前付きの優先順位クラスが見つからない場合、Pod は拒否されます。

2.6.2. Pod のプリエンブションについて

開発者が Pod を作成する場合、Pod はキューに入れられます。開発者が Pod の優先順位またはプリエンブションを設定している場合、スケジューラーはキューから Pod を選択し、Pod をノードにスケジュールしようとします。スケジューラーが Pod について指定されたすべての要件を満たす適切なノードに領域を見つけられない場合、プリエンブションロジックが保留中の Pod についてトリガーされます。

スケジューラーがノードで1つ以上の Pod のプリエンブションを実行する場合、優先順位の高い Pod 仕様の `nominatedNodeName` フィールドは、`nodename` フィールドと共にノードの名前に設定されます。スケジューラーは `nominatedNodeName` フィールドを使用して Pod の予約されたリソースを追跡し、またクラスターのプリエンブションについての情報をユーザーに提供します。

スケジューラーが優先順位の低い Pod のプリエンブションを実行した後に、スケジューラーは Pod の正常な終了期間を許可します。スケジューラーが優先順位の低い Pod の終了を待機する間に別のノードが利用可能になると、スケジューラーはそのノードに優先順位の高い Pod をスケジュールできます。その結果、Pod 仕様の `nominatedNodeName` フィールドおよび `nodeName` フィールドが異なる可能性があります。

さらに、スケジューラーがノード上で Pod のプリエンブションを実行し、終了を待機している場合で、保留中の Pod よりも優先順位の高い Pod をスケジュールする必要がある場合、スケジューラーは代わりに優先順位の高い Pod をスケジュールできます。その場合、スケジューラーは保留中の Pod の `nominatedNodeName` をクリアし、その Pod を他のノードの対象とすることができます。

プリエンブションは、ノードから優先順位の低いすべての Pod を削除する訳ではありません。スケジューラーは、優先順位の低い Pod の一部を削除して保留中の Pod をスケジュールできます。

スケジューラーは、保留中の Pod をノードにスケジュールできる場合にのみ、Pod のプリエンブションを実行するノードを考慮します。

2.6.2.1. プリエンブションを実行しない優先順位クラス

プリエンブションポリシーが `Never` に設定された Pod は優先順位の低い Pod よりも前のスケジューリングキューに置かれますが、他の Pod のプリエンブションを実行することはできません。スケジュールを待機しているプリエンブションを実行しない Pod は、十分なリソースが解放され、これがスケジュールされるまでスケジュールキュー内に留まります。他の Pod などのプリエンブションを実行しない Pod はスケジューラーのバックオフの対象になります。つまり、スケジューラーがこれらの Pod のスケジュールの試行に成功しない場合、低頻度で再試行されるため、優先順位の低い他の Pod をそれらの Pod よりも前にスケジュールできます。

プリエンブションを実行しない Pod については、他の優先順位の高い Pod が依然としてプリエンブションを実行できます。

2.6.2.2. Pod プリエンブションおよび他のスケジューラーの設定

Pod の優先順位およびプリエンブションを有効にする場合、他のスケジューラー設定を考慮します。

Pod の優先順位および Pod の Disruption Budget (停止状態の予算)

Pod の Disruption Budget (停止状態の予算) は一度に稼働している必要のあるレプリカの最小数またはパーセンテージを指定します。Pod の Disruption Budget (停止状態の予算) を指定する場合、Red Hat OpenShift Service on AWS は、Best Effort レベルで Pod のプリエンブションを実行する際にそれらを適用します。スケジューラーは、Pod の Disruption Budget (停止状態の予算) に違反しない範囲で Pod のプリエンブションを試行します。該当する Pod が見つからない場合には、Pod の Disruption Budget (停止状態の予算) の要件を無視して優先順位の低い Pod のプリエンブションが実行される可能性があります。

Pod の優先順位およびアフィニティー

Pod のアフィニティーは、新規 Pod が同じラベルを持つ他の Pod と同じノードにスケジュールされることを要求します。

保留中の Pod にノード上の 1 つ以上の優先順位の低い Pod との Pod 間のアフィニティーがある場合、スケジューラーはアフィニティーの要件を違反せずに優先順位の低い Pod のプリエンブションを実行することはできません。この場合、スケジューラーは保留中の Pod をスケジュールするための別のノードを探します。ただし、スケジューラーが適切なノードを見つけることは保証できず、保留中の Pod がスケジュールされない可能性があります。

この状態を防ぐには、優先順位が等しい Pod との Pod のアフィニティーの設定を慎重に行ってください。

2.6.2.3. プリエンブションが実行された Pod の正常な終了

Pod のプリエンブションの実行中、スケジューラーは Pod の正常な終了期間が期限切れになるのを待機します。その後、Pod は機能を完了し、終了します。Pod がこの期間後も終了しない場合、スケジューラーは Pod を強制終了します。この正常な終了期間により、スケジューラーによる Pod のプリエンブションの実行時と保留中の Pod のノードへのスケジュール時に時間差が出ます。

この時間差を最小限にするには、優先順位の低い Pod の正常な終了期間を短く設定します。

2.6.3. 優先順位およびプリエンプレションの設定

Pod 仕様で `priorityClassName` を使用して優先順位クラスオブジェクトを作成し、Pod を優先順位に関連付けることで、Pod の優先度およびプリエンプレションを適用できます。



注記

優先クラスを既存のスケジュール済み Pod に直接追加することはできません。

手順

優先順位およびプリエンプレションを使用するようにクラスターを設定するには、以下を実行します。

1. 次のような YAML ファイルを作成して、優先順位クラスの名前を含む Pod 仕様を定義します。

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    env: test
spec:
  containers:
  - name: nginx
    image: nginx
    imagePullPolicy: IfNotPresent
    priorityClassName: system-cluster-critical ❶
```

❶

この Pod で使用する優先順位クラスを指定します。

2. Pod を作成します。

```
$ oc create -f <file-name>.yaml
```

優先順位の名前は Pod 設定または Pod テンプレートに直接追加できます。

2.7. ノードセクターの使用による特定ノードへの POD の配置

ノードセクターは、キーと値のペアのマップを指定します。ルールは、ノード上のカスタムラベルと Pod で指定されたセクターを使用して定義されます。

Pod がノードで実行する要件を満たすには、Pod はノードのラベルとして示されるキーと値のペアを持っている必要があります。

同じ Pod 設定でノードのアフィニティーとノードセクターを使用している場合、以下の重要な考慮事項を参照してください。

2.7.1. ノードセクターの使用による Pod 配置の制御

Pod でノードセクターを使用し、ノードでラベルを使用して、Pod がスケジュールされる場所を制御できます。ノードセクターを使用すると、Red Hat OpenShift Service on AWS は一致するラベルが含まれるノード上に Pod をスケジュールします。

ラベルをノード、コンピューティングマシンセット、またはマシン設定に追加します。コンピューティングマシンセットにラベルを追加すると、ノードまたはマシンが停止した場合に、新規ノードにそのラベルが追加されます。ノードまたはマシン設定に追加されるラベルは、ノードまたはマシンが停止すると維持されません。

ノードセクターを既存 Pod に追加するには、ノードセクターを ReplicaSet オブジェクト、DaemonSet オブジェクト、StatefulSet オブジェクト、Deployment オブジェクト、または DeploymentConfig オブジェクトなどの Pod の制御オブジェクトに追加します。制御オブジェクト下の既存 Pod は、一致するラベルを持つノードで再作成されます。新規 Pod を作成する場合、ノードセクターを Pod 仕様に直接追加できます。Pod に制御オブジェクトがない場合は、Pod を削除し、Pod 仕様を編集して、Pod を再作成する必要があります。



注記

ノードセクターを既存のスケジュールされている Pod に直接追加することはできません。

前提条件

ノードセクターを既存 Pod に追加するには、Pod の制御オブジェクトを判別します。たとえば、router-default-66d5cf9464-m2g75 Pod は router-default-66d5cf9464 レプリカセットによって制御されます。

```
$ oc describe pod router-default-66d5cf9464-7pwkc
```

出力例

```
kind: Pod
apiVersion: v1
metadata:
# ...
Name:          router-default-66d5cf9464-7pwkc
Namespace:     openshift-ingress
# ...
Controlled By: ReplicaSet/router-default-66d5cf9464
# ...
```

Web コンソールでは、Pod YAML の `ownerReferences` に制御オブジェクトをリスト表示します。

```
apiVersion: v1
kind: Pod
metadata:
  name: router-default-66d5cf9464-7pwkc
# ...
ownerReferences:
- apiVersion: apps/v1
  kind: ReplicaSet
  name: router-default-66d5cf9464
  uid: d81dd094-da26-11e9-a48a-128e7edf0312
  controller: true
  blockOwnerDeletion: true
# ...
```

手順

- 一致するノードセクターを Pod に追加します。
- ノードセクターを既存 Pod および新規 Pod に追加するには、ノードセクターを Pod の制御オブジェクトに追加します。

ラベルを含む ReplicaSet オブジェクトのサンプル

```
kind: ReplicaSet
apiVersion: apps/v1
metadata:
  name: hello-node-6fbccf8d9
# ...
spec:
# ...
template:
  metadata:
    creationTimestamp: null
  labels:
    ingresscontroller.operator.openshift.io/deployment-ingresscontroller: default
    pod-template-hash: 66d5cf9464
  spec:
    nodeSelector:
      kubernetes.io/os: linux
      node-role.kubernetes.io/worker: "
      type: user-node ①
# ...
```

①

ノードセレクターを追加します。

○

ノードセレクターを特定の新規 Pod に追加するには、セレクターを Pod オブジェクトに直接追加します。

ノードセレクターを持つ Pod オブジェクトの例

```
apiVersion: v1
kind: Pod
metadata:
  name: hello-node-6fbccf8d9
# ...
spec:
  nodeSelector:
    region: east
    type: user-node
# ...
```



注記

ノードセレクターを既存のスケジュールされている Pod に直接追加することはできません。

第3章 CUSTOM METRICS AUTOSCALER OPERATOR を使用した POD の自動スケーリング

3.1. CUSTOM METRICS AUTOSCALER OPERATOR の概要

開発者は、Red Hat OpenShift の Custom Metrics Autoscaler Operator を使用して、CPU やメモリー以外のもも含むカスタムメトリクスに基づいて、Red Hat OpenShift Service on AWS のデプロイメント、ステートフルセット、カスタムリソース、またはジョブの Pod 数を自動的に増減する方法を指定できます。

Custom Metrics Autoscaler Operator は、Kubernetes Event Driven Autoscaler (KEDA) に基づくオプションの Operator であり、Pod メトリクス以外の追加のメトリクスソースを使用してワークロードをスケーリングできます。

カスタムメトリクスオートスケーラーは現在、Prometheus、CPU、メモリー、および Apache Kafka メトリクスのみをサポートしています。

Custom Metrics Autoscaler Operator は、特定のアプリケーションからのカスタムの外部メトリクスに基づいて、Pod をスケールアップおよびスケールダウンします。他のアプリケーションは引き続き他のスケーリング方法を使用します。スケーラーとも呼ばれる トリガー を設定します。これは、カスタムメトリクスオートスケーラーがスケーリング方法を決定するために使用するイベントとメトリクスのソースです。カスタムメトリクスオートスケーラーはメトリクス API を使用して、外部メトリクスを Red Hat OpenShift Service on AWS が使用できる形式に変換します。カスタムメトリクスオートスケーラーは、実際のスケーリングを実行する Horizontal Pod Autoscaler (HPA) を作成します。

カスタムメトリクスオートスケーラーを使用するには、ワークロード用の ScaledObject または ScaledJob オブジェクトを作成します。これらは、スケーリングメタデータを定義するカスタムリソース (CR) です。スケーリングするデプロイメントまたはジョブ、スケーリングするメトリクスのソース (トリガー)、許可される最小および最大レプリカ数などのその他のパラメーターを指定します。



注記

スケーリングするワークロードごとに、スケーリングされたオブジェクトまたはスケーリングされたジョブを 1 つだけ作成できます。また、スケーリングされたオブジェクトまたはスケーリングされたジョブと Horizontal Pod Autoscaler (HPA) を同じワークロードで使用することはできません。

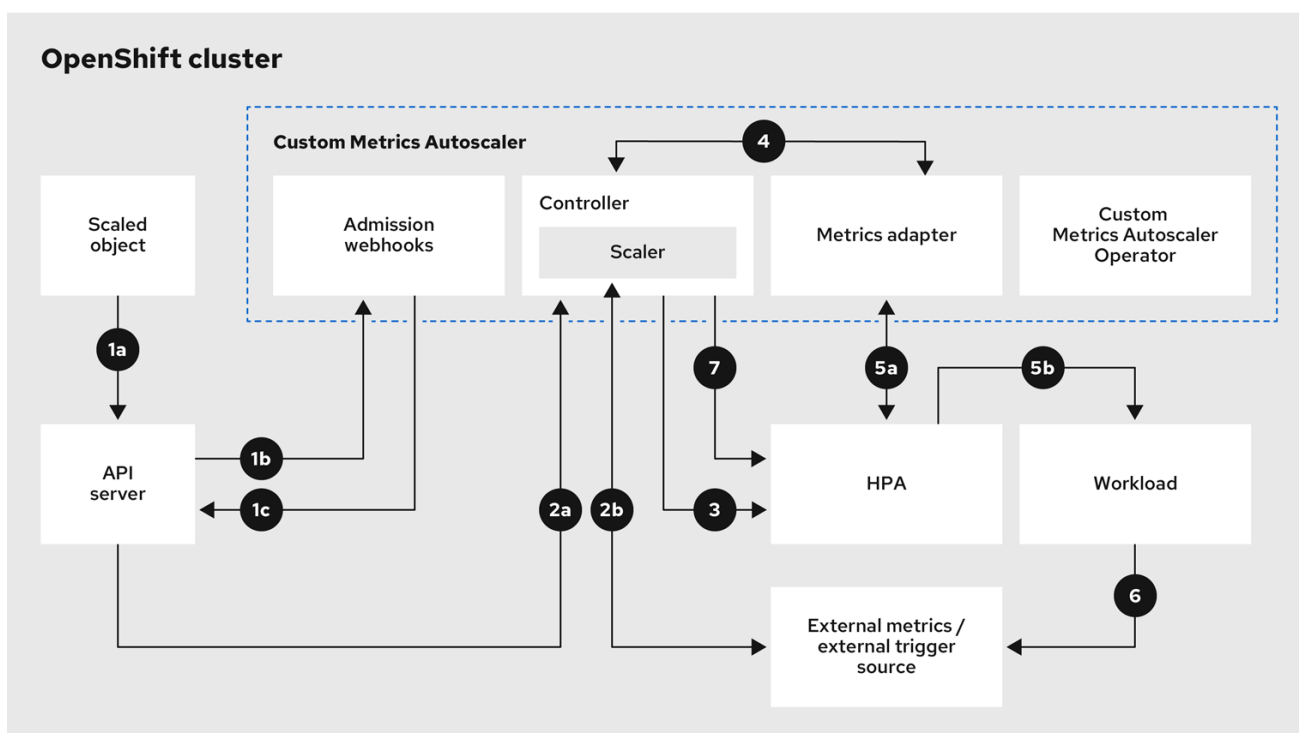
カスタムメトリクスオートスケーラーは、HPA とは異なり、ゼロにスケーリングできます。カスタムメトリクスオートスケーラー CR の minReplicaCount 値を 0 に設定すると、カスタムメトリクスオートスケーラーはワークロードを 1 レプリカから 0 レプリカにスケールダウンするか、0 レプリカから 1 にスケールアップします。これは、アクティベーションフェーズとして知られています。1 つのレ

プリカにスケールアップした後、HPA はスケーリングを制御します。これは スケーリングフェーズ として知られています。

一部のトリガーにより、クラスターメトリクスオートスケーラーによってスケーリングされるレプリカの数を変更できます。いずれの場合も、アクティベーションフェーズを設定するパラメーターは、`activation` で始まる同じフレーズを常に使用します。たとえば、`threshold` パラメーターがスケーリングを設定する場合、`activationThreshold` はアクティベーションを設定します。アクティベーションフェーズとスケーリングフェーズを設定すると、スケーリングポリシーの柔軟性が向上します。たとえば、アクティベーションフェーズをより高く設定することで、メトリクスが特に低い場合にスケールアップまたはスケールダウンを防ぐことができます。

それぞれ異なる決定を行う場合は、スケーリングの値よりもアクティベーションの値が優先されます。たとえば、`threshold` が 10 に設定されていて、`activationThreshold` が 50 である場合にメトリクスが 40 を報告した場合、スケーラーはアクティブにならず、HPA が 4 つのインスタンスを必要とする場合でも Pod はゼロにスケーリングされます。

図3.1 カスタムメトリクスオートスケーラーのワークフロー



565_OpenShift_0224

1. クラスタ上のワークロード用のスケーリングされたオブジェクトのカスタムリソースを作成または変更します。オブジェクトには、そのワークロードのスケーリング設定を含めます。OpenShift API サーバーは、新しいオブジェクトを受け入れる前に、そのオブジェクトをカスタムメトリクスオートスケーラーのアドミッション Webhook プロセスに送信して、オブジェクトが有効であることを確認します。検証が成功すると、API サーバーはオブジェクトを永続化します。

2.

カスタムメトリクスオートスケーラーコントローラーが、スケーリングされたオブジェクトの更新または変更を監視します。OpenShift API サーバーがコントローラーに変更を通知すると、コントローラーは、オブジェクト内で指定されている外部トリガーソース (データソースとも呼ばれる) を監視して、メトリクスデータの変更を確認します。1 つ以上のスケーラーが外部トリガーソースからのスケーリングデータを要求します。たとえば、Kafka トリガータイプの場合、コントローラーは Kafka スケーラーを使用して Kafka インスタンスと通信し、トリガーによって要求されたデータを取得します。

3. コントローラーが、スケーリングされたオブジェクトの水平 Pod オートスケーラーオブジェクトを作成します。その結果、Horizontal Pod Autoscaler (HPA) Operator が、トリガーに関連付けられたスケーリングデータの監視を開始します。HPA は、クラスターの OpenShift API サーバーエンドポイントからスケーリングデータを要求します。
4. OpenShift API サーバーエンドポイントが、カスタムメトリクスオートスケーラーのメトリクスアダプターによって提供されます。メトリクスアダプターは、カスタムメトリクスの要求を受信すると、コントローラーへの GRPC 接続を使用して、スケーラーから受信した最新のトリガーデータを要求します。
5. HPA がメトリクスアダプターから受信したデータに基づいてスケーリングを決定し、レプリカを増減することでワークロードをスケールアップまたはスケールダウンします。
6. 運用中に、ワークロードがスケーリングメトリクスに影響を与えることがあります。たとえば、Kafka キュー内の作業を処理するためにワークロードがスケールアップされた場合、ワークロードがすべての作業を処理した後、キューのサイズが減少します。その結果、ワークロードがスケールダウンされます。
7. メトリクスが `minReplicaCount` 値で指定された範囲内にある場合、カスタムメトリクスオートスケーラーコントローラーがすべてのスケーリングを無効にして、レプリカ数を一定に維持します。メトリクスがその範囲を超える場合、カスタムメトリクスオートスケーラーコントローラーはスケーリングを有効にして、HPA がワークロードをスケーリングできるようにします。スケーリングが無効になっている間、HPA は何もアクションを実行しません。

3.2. CUSTOM METRICS AUTOSCALER OPERATOR リリースノート

Red Hat OpenShift の Custom Metrics Autoscaler Operator のリリースノートでは、新機能および拡張機能、非推奨となった機能、および既知の問題について説明しています。

Custom Metrics Autoscaler Operator は、Kubernetes ベースの Event Driven Autoscaler (KEDA) を使用し、Red Hat OpenShift Service on AWS の Horizontal Pod Autoscaler (HPA) の上に構築されます。

**注記**

Red Hat OpenShift の Custom Metrics Autoscaler Operator のロギングサブシステムは、インストール可能なコンポーネントとして提供され、コアの Red Hat OpenShift Service on AWS とは異なるリリースサイクルを備えています。[Red Hat OpenShift Container Platform ライフサイクルポリシー](#) はリリースの互換性を概説しています。

3.2.1. サポート対象バージョン

次の表は、Red Hat OpenShift Service on AWS の各バージョンの Custom Metrics Autoscaler Operator バージョンを定義しています。

| バージョン | Red Hat OpenShift Service on AWS のバージョン | 一般公開 |
|--------|---|------|
| 2.11.2 | 4.15 | 一般公開 |
| 2.11.2 | 4.14 | 一般公開 |
| 2.11.2 | 4.13 | 一般公開 |
| 2.11.2 | 4.12 | 一般公開 |

3.2.2. Custom Metrics Autoscaler Operator 2.11.2-322 リリースノート

この Custom Metrics Autoscaler Operator 2.11.2-322 リリースでは、Red Hat OpenShift Service on AWS クラスタで Operator を実行するためのセキュリティー更新とバグ修正を使用できます。[RHSA-2023:6144](#) については、次のアドバイザリーを利用できます。

**重要**

このバージョンの Custom Metrics Autoscaler Operator をインストールする前に、以前にインストールされたテクノロジープレビューバージョンまたはコミュニティーがサポートするバージョンの KEDA を削除します。

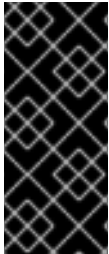
3.2.2.1. バグ修正

-

Custom Metrics Autoscaler Operator バージョン 3.11.2-311 は、Operator デプロイメントに必要なボリュームマウントなしにリリースされたため、Custom Metrics Autoscaler Operator Pod は 15 分ごとに再起動しました。この修正により、必要なボリュームマウントが Operator デプロイメントに追加されました。その結果、Operator は 15 分ごとに再起動なくなりました。(OCPBUGS-22361)

3.2.3. Custom Metrics Autoscaler Operator 2.11.2-311 リリースノート

この Custom Metrics Autoscaler Operator 2.11.2-311 リリースでは、Red Hat OpenShift Service on AWS クラスタで Operator を実行するための新機能とバグ修正を使用できます。Custom Metrics Autoscaler Operator 2.11.2-311 のコンポーネントは [RHBA-2023:5981](#) でリリースされました。



重要

このバージョンの Custom Metrics Autoscaler Operator をインストールする前に、以前にインストールされたテクノロジープレビューバージョンまたはコミュニティがサポートするバージョンの KEDA を削除します。

3.2.3.1. 新機能および機能拡張

3.2.3.1.1. Red Hat OpenShift Service on AWS (ROSA) と OpenShift Dedicated がサポートされるようになりました

Custom Metrics Autoscaler Operator 2.11.2-311 は、OpenShift ROSA および OpenShift Dedicated マネージドクラスタにインストールできます。Custom Metrics Autoscaler Operator の以前のバージョンは、openshift-keda namespace にのみインストールできました。これにより、Operator を OpenShift ROSA クラスタおよび OpenShift Dedicated クラスタにインストールできなくなりました。このバージョンの Custom Metrics Autoscaler では、openshift-operators や keda などの他の namespace へのインストールが可能になり、ROSA クラスタや Dedicated クラスタへのインストールが可能になります。

3.2.3.2. バグ修正

- 以前は、Custom Metrics Autoscaler Operator がインストールおよび設定されているが使用されていない場合、OpenShift CLI では、oc コマンドを入力すると、`couldn't get resource list for external.metrics.k8s.io/v1beta1: Got empty response for: external.metrics.k8s.io/v1beta1` エラーが報告されていました。このメッセージは無害ではありますが、混乱を引き起こす可能性がありました。この修正により、`Got empty response for: external.metrics...` エラーが不適切に表示されなくなりました。([OCPBUGS-15779](#))
- 以前は、設定変更後など、Keda Controller が変更されるたびに、Custom Metrics Autoscaler Operator によって管理されるオブジェクトに対するアノテーションやラベルの変更は、Custom Metrics Autoscaler Operator によって元に戻されました。これにより、オブジェクト内のラベルが継続的に変更されてしまいました。Custom Metrics Autoscaler は、独自のアノテーションを使用してラベルとアノテーションを管理するようになり、アノテーションやラベルが不適切に元に戻ることがなくなりました。([OCPBUGS-15590](#))

3.2.4. Custom Metrics Autoscaler Operator 2.10.1-267 リリースノート

この Custom Metrics Autoscaler Operator 2.10.1-267 リリースでは、Red Hat OpenShift Service on AWS クラスタで Operator を実行するための新機能とバグ修正を使用できます。Custom Metrics Autoscaler Operator 2.10.1-267 のコンポーネントは [RHBA-2023:4089](#) でリリースされました。



重要

このバージョンの Custom Metrics Autoscaler Operator をインストールする前に、以前にインストールされたテクノロジープレビューバージョンまたはコミュニティがサポートするバージョンの KEDA を削除します。

3.2.4.1. バグ修正

- 以前は、custom-metrics-autoscaler イメージと custom-metrics-autoscaler-adapter イメージにはタイムゾーン情報が含まれていませんでした。このため、コントローラーがタイムゾーン情報を見つけることができず、cron トリガーを使用してスケーリングされたオブジェクトが機能しませんでした。今回の修正により、イメージビルドにタイムゾーン情報が含まれるようになりました。その結果、cron トリガーを含むスケールされたオブジェクトが適切に機能するようになりました。([OCPBUGS-15264](#))
- 以前のバージョンでは、Custom Metrics Autoscaler Operator は、他の namespace 内のオブジェクトやクラスタースコープのオブジェクトを含む、すべてのマネージドオブジェクトの所有権を取得しようとしていました。このため、Custom Metrics Autoscaler Operator は API サーバーに必要な認証情報を読み取るためのロールバインディングを作成できませんでした。これにより、kube-system namespace でエラーが発生しました。今回の修正により、Custom Metrics Autoscaler Operator は、別の namespace 内のオブジェクトまたはクラスタースコープのオブジェクトへの ownerReference フィールドの追加をスキップします。その結果、ロールバインディングがエラーなしで作成されるようになりました。([OCPBUGS-15038](#))
- 以前のバージョンでは、Custom Metrics Autoscaler Operator は、ownshift-keda namespace に ownerReferences フィールドを追加しました。これによって機能上の問題が発生することはありませんでしたが、このフィールドの存在によりクラスター管理者が混乱する可能性があります。今回の修正により、Custom Metrics Autoscaler Operator は ownerReference フィールドを openshift-keda namespace に追加しなくなりました。その結果、openshift-keda namespace には余分な ownerReference フィールドが含まれなくなりました。([OCPBUGS-15293](#))
- 以前のバージョンでは、Pod ID 以外の認証方法で設定された Prometheus トリガーを使用し、podIdentity パラメーターが none に設定されている場合、トリガーはスケーリングに失敗しました。今回の修正により、OpenShift の Custom Metrics Autoscaler は、Pod ID プロバイダタイプ none を適切に処理できるようになりました。その結果、Pod ID 以外の認証方法で設定され、podIdentity パラメーターが none に設定された Prometheus トリガーが適切にスケーリングされるようになりました。([OCPBUGS-15274](#))

3.2.5. Custom Metrics Autoscaler Operator 2.10.1 リリースノート

この Custom Metrics Autoscaler Operator 2.10.1 リリースでは、Red Hat OpenShift Service on AWS クラスタで Operator を実行するための新機能とバグ修正を使用できます。Custom Metrics Autoscaler Operator 2.10.1 のコンポーネントは [RHEA-2023:3199](#) でリリースされました。



重要

このバージョンの Custom Metrics Autoscaler Operator をインストールする前に、以前にインストールされたテクノロジープレビューバージョンまたはコミュニティがサポートするバージョンの KEDA を削除します。

3.2.5.1. 新機能および機能拡張

3.2.5.1.1. Custom Metrics Autoscaler Operator の一般提供

Custom Metrics Autoscaler Operator バージョン 2.10.1 以降で、Custom Metrics Autoscaler Operator の一般提供が開始されました。



重要

スケーリングされたジョブを使用したスケーリングはテクノロジープレビュー機能です。テクノロジープレビュー機能は、Red Hat 製品サポートのサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではない場合があります。Red Hat は、実稼働環境でこれらを使用することを推奨していません。テクノロジープレビュー機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、[テクノロジープレビュー機能のサポート範囲](#) を参照してください。

3.2.5.1.2. パフォーマンスメトリクス

Prometheus Query Language (PromQL) を使用して、Custom Metrics Autoscaler Operator でメトリクスのクエリーを行えるようになりました。

3.2.5.1.3. スケーリングされたオブジェクトのカスタムメトリクス自動スケーリングの一時停止

必要に応じてスケーリングされたオブジェクトの自動スケーリングを一時停止し、準備ができたら再開できるようになりました。

3.2.5.1.4. スケーリングされたオブジェクトのレプリカフォールバック

スケーリングされたオブジェクトがソースからメトリクスを取得できなかった場合に、フォールバックするレプリカの数指定できるようになりました。

3.2.5.1.5. スケーリングされたオブジェクトのカスタマイズ可能な HPA 命名

スケーリングされたオブジェクトで、水平 Pod オートスケーラーのカスタム名を指定できるようになりました。

3.2.5.1.6. アクティブ化およびスケーリングのしきい値

水平 Pod オートスケーラー (HPA) は 0 レプリカへの、または 0 レプリカからのスケーリングができないため、Custom Metrics Autoscaler Operator がそのスケーリングを実行し、その後 HPA がスケーリングを実行します。レプリカの数に基づき HPA が自動スケーリングを引き継ぐタイミングを指定できるようになりました。これにより、スケーリングポリシーの柔軟性が向上します。

3.2.6. Custom Metrics Autoscaler Operator 2.8.2-174 リリースノート

この Custom Metrics Autoscaler Operator 2.8.2-174 リリースでは、Red Hat OpenShift Service on AWS クラスタで Operator を実行するための新機能とバグ修正を使用できます。Custom Metrics Autoscaler Operator 2.8.2-174 のコンポーネントは [RHEA-2023:1683](#) でリリースされました。



重要

Custom Metrics Autoscaler Operator バージョン 2.8.2-174 は、[テクノロジープレビュー](#) 機能です。

3.2.6.1. 新機能および機能拡張

3.2.6.1.1. Operator のアップグレードサポート

以前の Custom Metrics Autoscaler Operator バージョンからアップグレードできるようになりました。Operator のアップグレードについて、詳しくは「[関連情報](#)」の「Operator の更新チャンネルの変更」を参照してください。

3.2.6.1.2. must-gather サポート

Red Hat OpenShift Service on AWS の must-gather ツールを使用して、Custom Metrics Autoscaler Operator とそのコンポーネントに関するデータを収集できるようになりました。現時点

で、**Custom Metrics Autoscaler** で **must-gather** ツールを使用するプロセスは、他の **Operator** とは異なります。詳細は、「[関連情報](#)」の「[デバッグデータの収集](#)」を参照してください。

3.2.7. Custom Metrics Autoscaler Operator 2.8.2 リリースノート

この **Custom Metrics Autoscaler Operator 2.8.2** リリースでは、**Operator in an Red Hat OpenShift Service on AWS** クラスタで **Operator** を実行するための新機能とバグ修正を使用できます。**Custom Metrics Autoscaler Operator 2.8.2** のコンポーネントは [RHSA-2023:1042](#) でリリースされました。



重要

Custom Metrics Autoscaler Operator バージョン 2.8.2 は [テクノロジープレビュー](#) 機能です。

3.2.7.1. 新機能および機能拡張

3.2.7.1.1. 監査ロギング

Custom Metrics Autoscaler Operator とその関連コンポーネントの監査ログを収集して表示できるようになりました。監査ログは、システムに影響を与えた一連のアクティビティを個別のユーザー、管理者その他システムのコンポーネント別に記述したセキュリティー関連の時系列のレコードです。

3.2.7.1.2. Apache Kafka メトリクスに基づくアプリケーションのスケーリング

KEDA Apache kafka トリガー/スケーラーを使用して、**Apache Kafka** トピックに基づいてデプロイメントをスケーリングできるようになりました。

3.2.7.1.3. CPU メトリクスに基づくアプリケーションのスケーリング

KEDA CPU トリガー/スケーラーを使用して、**CPU** メトリクスに基づいてデプロイメントをスケーリングできるようになりました。

3.2.7.1.4. メモリーメトリクスに基づくアプリケーションのスケーリング

KEDA メモリートリガー/スケーラーを使用して、メモリーメトリクスに基づいてデプロイメントをスケーリングできるようになりました。

3.3. カスタムメトリクスオートスケーラーのインストール

Custom Metrics Autoscaler Operator は、Red Hat OpenShift Service on AWS Web コンソールを使用してインストールできます。

インストールにより、以下の 5 つの CRD が作成されます。

- **ClusterTriggerAuthentication**
- **KedaController**
- **ScaledJob**
- **ScaledObject**
- **TriggerAuthentication**

3.3.1. カスタムメトリクスオートスケーラーのインストール

次の手順を使用して、Custom Metrics Autoscaler Operator をインストールできます。

前提条件

- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。
- これまでにインストールしたテクノロジープレビューバージョンの **Cluster Metrics Autoscaler Operator** を削除する。
- コミュニティーベースの **KEDA** バージョンをすべて削除する。

次のコマンドを実行して、**KEDA 1.x** カスタムリソース定義を削除する。

```
$ oc delete crd scaledobjects.keda.k8s.io
```

```
$ oc delete crd triggerauthentications.keda.k8s.io
```

- **keda namespace が存在することを確認します。存在しない場合は、keda namespace を手動で作成する必要があります。**

手順

1. **Red Hat OpenShift Service on AWS Web コンソールで、Operator → OperatorHub をクリックします。**
2. **使用可能な Operator のリストから Custom Metrics Autoscaler を選択し、Install をクリックします。**
3. **Install Operator ページで、Installation Mode に A specific namespace on the cluster オプションが選択されていることを確認します。**
4. **Installed Namespace で、Select a namespace をクリックします。**
5. **Select Project をクリックします。**
 - **keda namespace が存在する場合は、リストから keda を選択します。**
 - **keda namespace が存在しない場合は、以下を実行します。**
 - a. **Create Project を選択して、Create Project ウィンドウを開きます。**
 - b. **Name フィールドに keda と入力します。**
 - c. **Display Name フィールドに、keda などのわかりやすい名前を入力します。**
 - d. **オプション: Display Name フィールドに、namespace の説明を追加します。**

- e. **Create** をクリックします。
6. **Install** をクリックします。
 7. **Custom Metrics Autoscaler Operator** コンポーネントをリスト表示して、インストールを確認します。
 - a. **Workloads** → **Pods** に移動します。
 - b. ドロップダウンメニューから **keda** プロジェクトを選択し、**custom-metrics-autoscaler-operator-*** Pod が実行されていることを確認します。
 - c. **Workloads** → **Deployments** に移動して、**custom-metrics-autoscaler-operator** デプロイメントが実行されていることを確認します。
 8. オプション: 次のコマンドを使用して、**OpenShift CLI** でインストールを確認します。

```
$ oc get all -n keda
```

以下のような出力が表示されます。

出力例

```
NAME                                READY STATUS RESTARTS AGE
pod/custom-metrics-autoscaler-operator-5fd8d9ffd8-xt4xp 1/1 Running 0
18m

NAME                                READY UP-TO-DATE AVAILABLE AGE
deployment.apps/custom-metrics-autoscaler-operator 1/1 1 1 18m

NAME                                DESIRED CURRENT READY AGE
replicaset.apps/custom-metrics-autoscaler-operator-5fd8d9ffd8 1 1 1
18m
```

9. 必要な CRD を作成する KedaController カスタムリソースをインストールします。
 - a. Red Hat OpenShift Service on AWS Web コンソールで、Operators → Installed Operators をクリックします。
 - b. Custom Metrics Autoscaler をクリックします。
 - c. Operator Details ページで、KedaController タブをクリックします。
 - d. KedaController タブで、Create KedaController をクリックしてファイルを編集します。

```

kind: KedaController
apiVersion: keda.sh/v1alpha1
metadata:
  name: keda
  namespace: keda
spec:
  watchNamespace: "" ①
  operator:
    logLevel: info ②
    logEncoder: console ③
  metricsServer:
    logLevel: '0' ④
    auditConfig: ⑤
    logFormat: "json"
    logOutputVolumeClaim: "persistentVolumeClaimName"
  policy:
    rules:
      - level: Metadata
    omitStages: ["RequestReceived"]
    omitManagedFields: false
  lifetime:
    maxAge: "2"
    maxBackup: "1"
    maxSize: "50"
  serviceAccount: {}

```

①

Custom Metrics Autoscaler Operator がアプリケーションをスケーリングする単一の namespace を指定します。空白のままにするか、または空にして、すべての namespace でアプリケーションをスケーリングします。このフィールドは、namespace があるか、空である必要があります。デフォルト値は空です。

2

Custom Metrics Autoscaler Operator ログメッセージの詳細レベルを指定します。許可される値は debug、info、error です。デフォルトは info です。

3

Custom Metrics Autoscaler Operator ログメッセージのログ形式を指定します。許可される値は console または json です。デフォルトは コンソール です。

4

Custom Metrics Autoscaler Metrics Server のログレベルを指定します。許可される値は、info の場合は 0 で、debug の場合は 4 です。デフォルトは 0 です。

5

Custom Metrics Autoscaler Operator の監査ログをアクティブにして、使用する監査ポリシーを指定します (「監査ログの設定」セクションを参照)。

e.

Create をクリックして KEDA コントローラーを作成します。

3.4. カスタムメトリクスオートスケーラートリガーについて

スケーラーとも呼ばれるトリガーは、Custom Metrics Autoscaler Operator が Pod をスケーリングするために使用するメトリクスを提供します。

カスタムメトリクスオートスケーラーは現在、Prometheus、CPU、メモリー、および Apache Kafka トリガーのみをサポートしています。

以下のセクションで説明するように、ScaledObject または ScaledJob カスタムリソースを使用して、特定のオブジェクトのトリガーを設定します。

3.4.1. Prometheus トリガーについて

Prometheus メトリクスに基づいて Pod をスケーリングできます。このメトリクスは、インストール済みの Red Hat OpenShift Service on AWS モニタリングまたは外部 Prometheus サーバーをメトリクスソースとして使用できます。Red Hat OpenShift Service on AWS モニタリングをメトリクスのソースとして使用するために必要な設定は、「関連情報」を参照してください。



注記

カスタムメトリクスオートスケーラーがスケーリングしているアプリケーションから Prometheus がメトリクスを収集している場合は、カスタムリソースで最小レプリカ数を 0 に設定しないでください。アプリケーション Pod がないと、カスタムメトリクスオートスケーラーにスケーリングの基準となるメトリクスが提供されません。

Prometheus ターゲットを使用したスケーリングされたオブジェクトの例

```

apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  name: prom-scaledobject
  namespace: my-namespace
spec:
  # ...
  triggers:
  - type: prometheus ①
    metadata:
      serverAddress: https://thanos-querier.openshift-monitoring.svc.cluster.local:9092 ②
      namespace: kedatest ③
      metricName: http_requests_total ④
      threshold: '5' ⑤
      query: sum(rate(http_requests_total{job="test-app"}[1m])) ⑥
      authModes: basic ⑦
      cortexOrgID: my-org ⑧
      ignoreNullValues: "false" ⑨
      unsafeSsl: "false" ⑩

```

①

Prometheus をトリガータイプとして指定します。

②

Prometheus サーバーのアドレスを指定します。この例では、Red Hat OpenShift Service on AWS モニタリングを使用します。

③

オプション: スケーリングするオブジェクトの namespace を指定します。メトリクスのソースとして Red Hat OpenShift Service on AWS モニタリングを使用する場合、このパラメーターは必須です。

4

`external.metrics.k8s.io` API でメトリクスを識別する名前を指定します。複数のトリガーを使用している場合、すべてのメトリクス名が一意である必要があります。

5

スケーリングをトリガーする値を指定します。引用符で囲まれた文字列値として指定する必要があります。

6

使用する Prometheus クエリーを指定します。

7

使用する認証方法を指定します。Prometheus スケーラーは、ベアラー認証 (bearer)、基本認証 (basic)、または TLS 認証 (tls) をサポートしています。以下のセクションで説明するように、トリガー認証で特定の認証パラメーターを設定します。必要に応じて、シークレットを使用することもできます。

8

オプション: `X-Scope-OrgID` ヘッダーを Prometheus のマルチテナント [Cortex](#) または [Mimir](#) ストレージに渡します。このパラメーターは、Prometheus が返す必要のあるデータを示すために、マルチテナント Prometheus ストレージでのみ必要です。

9

オプション: Prometheus ターゲットが失われた場合のトリガーの処理方法を指定します。

- `true` の場合、Prometheus ターゲットが失われても、トリガーは動作し続けます。これがデフォルトの動作です。
- `false` の場合、Prometheus ターゲットが失われると、トリガーはエラーを返しません。

10

オプション: 証明書チェックをスキップするかどうかを指定します。たとえば、Prometheus エンドポイントで自己署名証明書を使用する場合は、チェックをスキップできます。

- true の場合、証明書チェックが実行されます。
- false の場合、証明書チェックは実行されません。これがデフォルトの動作です。

3.4.1.1. Red Hat OpenShift Service on AWS モニタリングを使用するためのカスタムメトリクスオートスケーラーの設定

カスタムメトリクスオートスケーラーが使用するメトリクスのソースとして、インストール済みの **Red Hat OpenShift Service on AWS Prometheus** モニタリングを使用できます。ただし、実行する必要がある追加の設定がいくつかあります。



注記

これらの手順は、外部 Prometheus ソースには必要ありません。

このセクションで説明するように、次のタスクを実行する必要があります。

- トークンを取得するためのサービスアカウントを作成します。
- ロールを作成します。
- そのロールをサービスアカウントに追加します。
- Prometheus が使用するトリガー認証オブジェクトでトークンを参照します。

前提条件

- Red Hat OpenShift Service on AWS モニタリングをインストールしている。
- ユーザー定義のワークロードのモニタリングを、Red Hat OpenShift Service on AWS モニタリングで有効にしている (ユーザー定義のワークロードモニタリング設定マップの作成 セクションで説明)。

- Custom Metrics Autoscaler Operator をインストールしている。

手順

1. スケーリングするオブジェクトを含むプロジェクトに変更します。

```
$ oc project my-project
```

2. クラスタにサービスアカウントがない場合は、次のコマンドを使用してサービスアカウントを作成します。

```
$ oc create serviceaccount <service_account>
```

ここでは、以下のようになります。

```
<service_account>
```

サービスアカウントの名前を指定します。

3. 次のコマンドを使用して、サービスアカウントに割り当てられたトークンを見つけます。

```
$ oc describe serviceaccount <service_account>
```

ここでは、以下のようになります。

```
<service_account>
```

サービスアカウントの名前を指定します。

出力例

```
Name:          thanos
Namespace:     my-project
Labels:        <none>
Annotations:   <none>
Image pull secrets: thanos-dockercfg-nnwgj
```

```
Mountable secrets: thanos-dockercfg-nnwgj
```

```
Tokens:          thanos-token-9g4n5 1
```

```
Events:         <none>
```

1

トリガー認証でこのトークンを使用します。

4.

サービスアカウントトークンを使用してトリガー認証を作成します。

a.

以下のような YAML ファイルを作成します。

```
apiVersion: keda.sh/v1alpha1
kind: TriggerAuthentication
metadata:
  name: keda-trigger-auth-prometheus
spec:
  secretTargetRef: 1
  - parameter: bearerToken 2
    name: thanos-token-9g4n5 3
    key: token 4
  - parameter: ca
    name: thanos-token-9g4n5
    key: ca.crt
```

1

このオブジェクトが承認にシークレットを使用することを指定します。

2

トークンを使用して提供する認証パラメーターを指定します。

3

使用するトークンの名前を指定します。

4

指定されたパラメーターで使用するトークン内のキーを指定します。

- b. CR オブジェクトを作成します。

```
$ oc create -f <file-name>.yaml
```

5. Thanos メトリクスを読み取るためのロールを作成します。

- a. 次のパラメーターを使用して YAML ファイルを作成します。

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: thanos-metrics-reader
rules:
- apiGroups:
  - ""
  resources:
  - pods
  verbs:
  - get
- apiGroups:
  - metrics.k8s.io
  resources:
  - pods
  - nodes
  verbs:
  - get
  - list
  - watch
```

- b. CR オブジェクトを作成します。

```
$ oc create -f <file-name>.yaml
```

6. Thanos メトリクスを読み取るためのロールバインディングを作成します。

- a. 以下のような YAML ファイルを作成します。

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: thanos-metrics-reader 1
  namespace: my-project 2
roleRef:
```

```
apiGroup: rbac.authorization.k8s.io
kind: Role
name: thanos-metrics-reader
subjects:
- kind: ServiceAccount
  name: thanos ③
  namespace: my-project ④
```

①

作成したロールの名前を指定します。

②

スケーリングするオブジェクトの namespace を指定します。

③

ロールにバインドするサービスアカウントの名前を指定します。

④

スケーリングするオブジェクトの namespace を指定します。

b.

CR オブジェクトを作成します。

```
$ oc create -f <file-name>.yaml
```

「カスタムメトリクスオートスケーラーの追加方法について」で説明されているとおり、スケーリングされたオブジェクトまたはスケーリングされたジョブをデプロイして、アプリケーションの自動スケーリングを有効化できます。Red Hat OpenShift Service on AWS モニタリングをソースとして使用するには、トリガーまたはスケーラーに以下のパラメーターを含める必要があります。

- `triggers.type` は `prometheus` にしてください。
- `triggers.metadata.serverAddress` は `https://thanos-querier.openshift-monitoring.svc.cluster.local:9092` にしてください。
- `triggers.metadata.authModes` は `bearer` にしてください。
-

`triggers.metadata.namespace` は、スケーリングするオブジェクトの `namespace` に設定してください。

- `triggers.authenticationRef` は、直前の手順で指定されたトリガー認証リソースを指す必要があります。

3.4.2. CPU トリガーについて

CPU メトリクスに基づいて Pod をスケーリングできます。このトリガーは、クラスターメトリクスをメトリクスのソースとして使用します。

カスタムメトリクスオートスケーラーは、オブジェクトに関連付けられた Pod をスケーリングして、指定された CPU 使用率を維持します。オートスケーラーは、すべての Pod で指定された CPU 使用率を維持するために、最小数と最大数の間でレプリカ数を増減します。メモリートリガーは、Pod 全体のメモリー使用率を考慮します。Pod に複数のコンテナがある場合、メモリートリガーは Pod 内にあるすべてのコンテナの合計メモリー使用率を考慮します。



注記

- このトリガーは、`ScaledJob` カスタムリソースでは使用できません。
- メモリートリガーを使用してオブジェクトをスケーリングすると、複数のトリガーを使用している場合でも、オブジェクトは 0 にスケーリングされません。

CPU ターゲットを使用してスケーリングされたオブジェクトの例

```
apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  name: cpu-scaledobject
  namespace: my-namespace
spec:
  # ...
  triggers:
  - type: cpu ①
    metricType: Utilization ②
    metadata:
      value: '60' ③
    minReplicaCount: 1 ④
```

1

トリガータイプとして CPU を指定します。

2

使用するメトリクスのタイプ (Utilization または AverageValue のいずれか) を指定します。

3

スケーリングをトリガーする値を指定します。引用符で囲まれた文字列値として指定する必要があります。

- Utilization を使用する場合、ターゲット値は、関連する全 Pod のリソースメトリクスの平均値であり、Pod のリソースの要求値に占めるパーセンテージとして表されます。
- AverageValue を使用する場合、ターゲット値は、関連する全 Pod のメトリクスの平均値です。

4

スケールダウン時のレプリカの最小数を指定します。CPU トリガーの場合は、1 以上の値を入力します。CPU メトリクスのみを使用している場合、HPA はゼロにスケールできないためです。

3.4.3. メモリートリガーについて

メモリーメトリクスに基づいて Pod をスケーリングできます。このトリガーは、クラスターメトリクスをメトリクスのソースとして使用します。

カスタムメトリクスオートスケーラーは、オブジェクトに関連付けられた Pod をスケーリングして、指定されたメモリー使用率を維持します。オートスケーラーは、すべての Pod で指定のメモリー使用率を維持するために、最小数と最大数の間でレプリカ数を増減します。メモリートリガーは、Pod 全体のメモリー使用率を考慮します。Pod に複数のコンテナがある場合、メモリー使用率はすべてのコンテナの合計になります。



注記

- このトリガーは、ScaledJob カスタムリソースでは使用できません。
- メモリートリガーを使用してオブジェクトをスケーリングすると、複数のトリガーを使用している場合でも、オブジェクトは 0 にスケーリングされません。

メモリーターゲットを使用してスケーリングされたオブジェクトの例

```

apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  name: memory-scaledobject
  namespace: my-namespace
spec:
  # ...
  triggers:
  - type: memory ①
    metricType: Utilization ②
    metadata:
      value: '60' ③
      containerName: api ④

```

①

トリガータイプとしてメモリーを指定します。

②

使用するメトリクスのタイプ (Utilization または AverageValue のいずれか) を指定します。

③

スケーリングをトリガーする値を指定します。引用符で囲まれた文字列値として指定する必要があります。

•

Utilization を使用する場合、ターゲット値は、関連する全 Pod のリソースメトリクスの平均値であり、Pod のリソースの要求値に占めるパーセンテージとして表されます。

- **AverageValue** を使用する場合、ターゲット値は、関連する全 Pod のメトリクスの平均値です。

4

オプション: Pod 全体ではなく、そのコンテナのみのメモリー使用率に基づいて、スケーリングする個々のコンテナを指定します。この例では、`api` という名前のコンテナのみがスケーリングされます。

3.4.4. Kafka トリガーについて

Apache Kafka トピックまたは Kafka プロトコルをサポートするその他のサービスに基づいて Pod をスケーリングできます。カスタムメトリクスオートスケーラーは、スケーリングされるオブジェクトまたはスケーリングされるジョブで `allowIdleConsumers` パラメーターを `true` に設定しない限り、Kafka パーティションの数を超えてスケーリングしません。

注記

コンシューマーグループの数がトピック内のパーティションの数を超えると、余分なコンシューマーグループはそのままアイドル状態になります。これを回避するために、デフォルトではレプリカ数は次の値を超えません。

- トピックのパーティションの数 (トピックが指定されている場合)。
- コンシューマーグループ内の全トピックのパーティション数 (トピックが指定されていない場合)。
- スケーリングされるオブジェクトまたはスケーリングされるジョブの CR で指定された `maxReplicaCount`。

これらのデフォルトの動作は、`allowIdleConsumers` パラメーターを使用して無効にすることができます。

Kafka ターゲットを使用してスケーリングされたオブジェクトの例

apiVersion: keda.sh/v1alpha1


```
kind: ScaledObject
metadata:
  name: kafka-scaledobject
  namespace: my-namespace
spec:
  # ...
  triggers:
  - type: kafka ①
    metadata:
      topic: my-topic ②
      bootstrapServers: my-cluster-kafka-bootstrap.openshift-operators.svc:9092 ③
      consumerGroup: my-group ④
      lagThreshold: '10' ⑤
      activationLagThreshold: '5' ⑥
      offsetResetPolicy: latest ⑦
      allowIdleConsumers: true ⑧
      scaleToZeroOnInvalidOffset: false ⑨
      excludePersistentLag: false ⑩
      version: '1.0.0' ⑪
      partitionLimitation: '1,2,10-20,31' ⑫
```

①

トリガータイプとして Kafka を指定します。

②

Kafka がオフセットラグを処理している Kafka トピックの名前を指定します。

③

接続する Kafka ブローカーのコンマ区切りリストを指定します。

④

トピックのオフセットの確認と、関連するラグの処理に使用される Kafka コンシューマーグループの名前を指定します。

⑤

オプション: スケーリングをトリガーする平均ターゲット値を指定します。引用符で囲まれた文字列値として指定する必要があります。デフォルトは 5 です。

⑥

オプション: アクティベーションフェーズのターゲット値を指定します。引用符で囲まれた文字列値として指定する必要があります。

7

オプション: Kafka コンシューマーの Kafka オフセットリセットポリシーを指定します。使用可能な値は `latest` および `earliest` です。デフォルトは `latest` です。

8

オプション: Kafka レプリカの数とトピックのパーティションの数を超えることを許可するかどうかを指定します。

- `true` の場合、Kafka レプリカの数とトピックのパーティションの数を超えることができます。これにより、Kafka コンシューマーがアイドル状態になることが許容されます。
- `false` の場合、Kafka レプリカの数とトピックのパーティションの数を超えることはできません。これはデフォルトになります。

9

Kafka パーティションに有効なオフセットがない場合のトリガーの動作を指定します。

- `true` の場合、そのパーティションのコンシューマーはゼロにスケールされます。
- `false` の場合、スケールはそのパーティションのために 1 つのコンシューマーを保持します。これはデフォルトになります。

10

オプション: 現在のオフセットが前のポーリングサイクルの現在のオフセットと同じであるパーティションのパーティションラグをトリガーに含めるか除外するかを指定します。

- `true` の場合、スケールはこれらのパーティションのパーティションラグを除外します。
- `false` の場合、すべてのパーティションのコンシューマーラグがすべてトリガーに含まれます。これはデフォルトになります。

11

オプション: Kafka ブローカーのバージョンを指定します。引用符で囲まれた文字列値として指定する必要があります。デフォルトは 1.0.0 です。

12

オプション: スケーリングのスコープを適用するパーティション ID のコンマ区切りリストを指定します。指定されている場合、ラグの計算時にリスト内の ID のみが考慮されます。引用符で囲まれた文字列値として指定する必要があります。デフォルトでは、すべてのパーティションが考慮されます。

3.5. カスタムメトリクスオートスケーラートリガー認証について

トリガー認証を使用すると、関連付けられたコンテナで使用できるスケーリングされたオブジェクトまたはスケーリングされたジョブに認証情報を含めることができます。トリガー認証を使用して、Red Hat OpenShift Service on AWS シークレット、プラットフォームネイティブの Pod 認証メカニズム、環境変数などを渡すことができます。

スケーリングするオブジェクトと同じ namespace に TriggerAuthentication オブジェクトを定義します。そのトリガー認証は、その namespace 内のオブジェクトによってのみ使用できます。

または、複数の namespace のオブジェクト間で認証情報を共有するには、すべての namespace で使用できる ClusterTriggerAuthentication オブジェクトを作成できます。

トリガー認証とクラスタートリガー認証は同じ設定を使用します。ただし、クラスタートリガー認証では、スケーリングされたオブジェクトの認証参照に追加の kind パラメーターが必要です。

シークレットを使用したトリガー認証の例

```
kind: TriggerAuthentication
apiVersion: keda.sh/v1alpha1
metadata:
  name: secret-triggerauthentication
  namespace: my-namespace ①
spec:
  secretTargetRef: ②
  - parameter: user-name ③
    name: my-secret ④
```

```
key: USER_NAME 5
- parameter: password
name: my-secret
key: USER_PASSWORD
```

1

スケーリングするオブジェクトの namespace を指定します。

2

このトリガー認証が承認にシークレットを使用することを指定します。

3

シークレットを使用して提供する認証パラメーターを指定します。

4

使用するシークレットの名前を指定します。

5

指定されたパラメーターで使用するシークレットのキーを指定します。

シークレットを使用したクラスタトリガー認証の例

```
kind: ClusterTriggerAuthentication
apiVersion: keda.sh/v1alpha1
metadata: 1
  name: secret-cluster-triggerauthentication
spec:
  secretTargetRef: 2
  - parameter: user-name 3
    name: secret-name 4
    key: USER_NAME 5
  - parameter: user-password
    name: secret-name
    key: USER_PASSWORD
```

1

クラスタトリガー認証では `namespace` が使用されないことに注意してください。

2

このトリガー認証が承認にシークレットを使用することを指定します。

3

シークレットを使用して提供する認証パラメーターを指定します。

4

使用するシークレットの名前を指定します。

5

指定されたパラメーターで使用するシークレットのキーを指定します。

トークンを使用したトリガー認証の例

```
kind: TriggerAuthentication
apiVersion: keda.sh/v1alpha1
metadata:
  name: token-triggerauthentication
  namespace: my-namespace 1
spec:
  secretTargetRef: 2
  - parameter: bearerToken 3
    name: my-token-2vzfq 4
    key: token 5
  - parameter: ca
    name: my-token-2vzfq
    key: ca.crt
```

1

スケーリングするオブジェクトの `namespace` を指定します。

2

3

4

使用するトークンの名前を指定します。

5

指定されたパラメーターで使用するトークン内のキーを指定します。

環境変数を使用したトリガー認証の例

```
kind: TriggerAuthentication
apiVersion: keda.sh/v1alpha1
metadata:
  name: env-var-triggerauthentication
  namespace: my-namespace 1
spec:
  env: 2
  - parameter: access_key 3
    name: ACCESS_KEY 4
  containerName: my-container 5
```

1

スケーリングするオブジェクトの namespace を指定します。

2

このトリガー認証が承認に環境変数を使用することを指定します。

3

この変数で設定するパラメーターを指定します。

4

環境変数の名前を指定します。

5

Pod 認証プロバイダーを使用したトリガー認証の例

```
kind: TriggerAuthentication
apiVersion: keda.sh/v1alpha1
metadata:
  name: pod-id-triggerauthentication
  namespace: my-namespace ①
spec:
  podIdentity: ②
  provider: aws-eks ③
```

1

スケーリングするオブジェクトの `namespace` を指定します。

2

このトリガー認証が承認にプラットフォームネイティブの Pod 認証方法を使用することを指定します。

3

Pod ID を指定します。サポートされている値は、`none`、`azure`、`aws-eks`、または `aws-kiam` です。デフォルト値は `none` です。

3.5.1. トリガー認証の使用

トリガー認証とクラスタトリガー認証は、カスタムリソースを使用して認証を作成し、スケーリングされたオブジェクトまたはスケーリングされたジョブへの参照を追加することで使用します。

前提条件

- **Custom Metrics Autoscaler Operator** をインストールしている。
- シークレットを使用している場合は、**Secret** オブジェクトが存在する必要があります。次

に例を示します。

シークレットの例

```
apiVersion: v1
kind: Secret
metadata:
  name: my-secret
data:
  user-name: <base64_USER_NAME>
  password: <base64_USER_PASSWORD>
```

手順

1. **TriggerAuthentication** または **ClusterTriggerAuthentication** オブジェクトを作成します。
 - a. オブジェクトを定義する **YAML** ファイルを作成します。

シークレットを使用したトリガー認証の例

```
kind: TriggerAuthentication
apiVersion: keda.sh/v1alpha1
metadata:
  name: prom-triggerauthentication
  namespace: my-namespace
spec:
  secretTargetRef:
    - parameter: user-name
      name: my-secret
      key: USER_NAME
    - parameter: password
      name: my-secret
      key: USER_PASSWORD
```


- b. **TriggerAuthentication** オブジェクトを作成します。

```
$ oc create -f <filename>.yaml
```

2. トリガー認証を使用する **ScaledObject** YAML ファイルを作成または編集します。

- a. 次のコマンドを実行して、オブジェクトを定義する **YAML** ファイルを作成します。

トリガー認証を使用したスケーリングされたオブジェクトの例

```
apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  name: scaledobject
  namespace: my-namespace
spec:
  scaleTargetRef:
    name: example-deployment
  maxReplicaCount: 100
  minReplicaCount: 0
  pollingInterval: 30
  triggers:
  - type: prometheus
    metadata:
      serverAddress: https://thanos-querier.openshift-
        monitoring.svc.cluster.local:9092
      namespace: kedatest # replace <NAMESPACE>
      metricName: http_requests_total
      threshold: '5'
      query: sum(rate(http_requests_total{job="test-app"}[1m]))
      authModes: "basic"
    authenticationRef:
      name: prom-triggerauthentication ①
      kind: TriggerAuthentication ②
```

①

トリガー認証オブジェクトの名前を指定します。

②

クラスタトリガー認証を使用したスケーリングされたオブジェクトの例

```
apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  name: scaledobject
  namespace: my-namespace
spec:
  scaleTargetRef:
    name: example-deployment
  maxReplicaCount: 100
  minReplicaCount: 0
  pollingInterval: 30
  triggers:
  - type: prometheus
    metadata:
      serverAddress: https://thanos-querier.openshift-
monitoring.svc.cluster.local:9092
      namespace: kedatest # replace <NAMESPACE>
      metricName: http_requests_total
      threshold: '5'
      query: sum(rate(http_requests_total{job="test-app"}[1m]))
      authModes: "basic"
    authenticationRef:
      name: prom-cluster-triggerauthentication ①
      kind: ClusterTriggerAuthentication ②
```

①

トリガー認証オブジェクトの名前を指定します。

②

ClusterTriggerAuthentication を指定します。

b.

次のコマンドを実行して、スケーリングされたオブジェクトを作成します。

```
$ oc apply -f <filename>
```

3.6. スケーリングされたオブジェクトのカスタムメトリクスオートスケーラーの一時停止

必要に応じて、ワークロードの自動スケーリングを一時停止および再開できます。

たとえば、クラスターのメンテナンスを実行する前に自動スケーリングを一時停止したり、ミッションクリティカルではないワークロードを削除してリソース不足を回避したりできます。

3.6.1. カスタムメトリクスオートスケーラーの一時停止

スケーリングされたオブジェクトの自動スケーリングを一時停止するには、そのスケーリングされたオブジェクトのカスタムメトリクスオートスケーラーに `autoscaling.keda.sh/paused-replicas` アノテーションを追加します。カスタムメトリクスオートスケーラーは、そのワークロードのレプリカを指定された値にスケーリングし、アノテーションが削除されるまで自動スケーリングを一時停止します。

```
apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  annotations:
    autoscaling.keda.sh/paused-replicas: "4"
# ...
```

手順

1. 次のコマンドを使用して、ワークロードの `ScaledObject` CR を編集します。

```
$ oc edit ScaledObject scaledobject
```

2. `autoscaling.keda.sh/paused-replicas` アノテーションに任意の値を追加します。

```
apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  annotations:
    autoscaling.keda.sh/paused-replicas: "4" ①
  creationTimestamp: "2023-02-08T14:41:01Z"
  generation: 1
  name: scaledobject
  namespace: my-project
  resourceVersion: '65729'
  uid: f5aec682-acdf-4232-a783-58b5b82f5dd0
```

①

Custom Metrics Autoscaler Operator がレプリカを指定された値にスケーリングし、自動スケーリングを停止するよう指定します。

3.6.2. スケーリングされたオブジェクトのカスタムメトリクスオートスケーラーの再開

一時停止されたカスタムメトリクスオートスケーラーを再開するには、その ScaledObject の `autoscaling.keda.sh/paused-replicas` アノテーションを削除します。

```
apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  annotations:
    autoscaling.keda.sh/paused-replicas: "4"
# ...
```

手順

1. 次のコマンドを使用して、ワークロードの ScaledObject CR を編集します。

```
$ oc edit ScaledObject scaledobject
```

2. `autoscaling.keda.sh/paused-replicas` アノテーションを削除します。

```
apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  annotations:
    autoscaling.keda.sh/paused-replicas: "4" 1
  creationTimestamp: "2023-02-08T14:41:01Z"
  generation: 1
  name: scaledobject
  namespace: my-project
  resourceVersion: '65729'
  uid: f5aec682-acdf-4232-a783-58b5b82f5dd0
```

1

このアノテーションを削除して、一時停止されたカスタムメトリクスオートスケーラーを再開します。

3.7. 監査ログの収集

システムに影響を与えた一連のアクティビティを個別のユーザー、管理者その他システムのコンポーネント別に記述したセキュリティ関連の時系列のレコードを提供する、監査ログを収集できます。

たとえば、監査ログは、自動スケーリングリクエストの送信元を理解するのに役立ちます。これは、ユーザーアプリケーションによる自動スケーリングリクエストによってバックエンドが過負荷になり、問題のあるアプリケーションを特定する必要がある場合に重要な情報です。

3.7.1. 監査ログの設定

KedaController カスタムリソースを編集することで、Custom Metrics Autoscaler Operator の監査を設定できます。ログは、KedaController CR の永続ボリューム要求を使用して保護されたボリューム上の監査ログファイルに送信されます。

前提条件

- Custom Metrics Autoscaler Operator をインストールしている。

手順

1. KedaController カスタムリソースを編集して、auditConfig スタンザを追加します。

```
kind: KedaController
apiVersion: keda.sh/v1alpha1
metadata:
  name: keda
  namespace: keda
spec:
# ...
  metricsServer:
# ...
  auditConfig:
    logFormat: "json" ①
    logOutputVolumeClaim: "pvc-audit-log" ②
    policy:
      rules: ③
      - level: Metadata
    omitStages: "RequestReceived" ④
    omitManagedFields: false ⑤
  lifetime: ⑥
    maxAge: "2"
    maxBackup: "1"
    maxSize: "50"
```

①

監査ログの出力形式を legacy または json のいずれかで指定します。

②

3

どのイベントを記録し、どのデータを含めるかを指定します。

- **None:** イベントをログに記録しません。
- **Metadata:** ユーザー、タイムスタンプなど、リクエストのメタデータのみをログに記録します。リクエストテキストと応答テキストはログに記録しないでください。これはデフォルトになります。
- **Request:** メタデータと要求テキストのみをログに記録しますが、応答テキストはログに記録しません。このオプションは、リソース以外の要求には適用されません。
- **RequestResponse:** イベントのメタデータ、要求テキスト、および応答テキストをログに記録します。このオプションは、リソース以外の要求には適用されません。

4

イベントを作成しないステージを指定します。

5

リクエストおよび応答本文のマネージドフィールドが API 監査ログに書き込まれないようにするかどうかを指定します。フィールドを省略する場合は `true`、フィールドを含める場合は `false` を指定します。

6

監査ログのサイズと有効期間を指定します。

- **maxAge:** ファイル名にエンコードされたタイムスタンプに基づく、監査ログファイルを保持する最大日数。
- **maxBackup:** 保持する監査ログファイルの最大数。すべての監査ログファイルを保持するには、0 に設定します。
-

maxSize: ローテーションされる前の監査ログファイルの最大サイズ (メガバイト単位)。

検証

1. 監査ログファイルを直接表示します。

- a. `keda-metrics-apiserver-*` Pod の名前を取得します。

```
oc get pod -n keda
```

出力例

```
NAME                                READY STATUS RESTARTS AGE
custom-metrics-autoscaler-operator-5cb44cd75d-9v4lv 1/1 Running 0 8m20s
keda-metrics-apiserver-65c7cc44fd-rrl4r           1/1 Running 0 2m55s
keda-operator-776cbb6768-zpj5b                    1/1 Running 0 2m55s
```

- b. 次のようなコマンドを使用して、ログデータを表示します。

```
$ oc logs keda-metrics-apiserver-<hash>|grep -i metadata 1
```

1

オプション: `grep` コマンドを使用して、表示するログレベル (Metadata、Request、RequestResponse) を指定できます。

以下に例を示します。

```
$ oc logs keda-metrics-apiserver-65c7cc44fd-rrl4r|grep -i metadata
```

出力例

```
...
{"kind":"Event","apiVersion":"audit.k8s.io/v1","level":"Metadata","auditID":"4c81d
41b-3dab-4675-90ce-
20b87ce24013","stage":"ResponseComplete","requestURI":"/healthz","verb":"get"
,"user":{"username":"system:anonymous","groups":
["system:unauthenticated"]},"sourceIPs":["10.131.0.1"],"userAgent":"kube-
probe/1.28","responseStatus":{"metadata":
{},"code":200},"requestReceivedTimestamp":"2023-02-
16T13:00:03.554567Z","stageTimestamp":"2023-02-
16T13:00:03.555032Z","annotations":
{"authorization.k8s.io/decision":"allow","authorization.k8s.io/reason":""}}
...
```

2.

または、特定のログを表示できます。

a.

次のようなコマンドを使用して、keda-metrics-apiserver-* Pod にログインします。

```
$ oc rsh pod/keda-metrics-apiserver-<hash> -n keda
```

以下に例を示します。

```
$ oc rsh pod/keda-metrics-apiserver-65c7cc44fd-rrl4r -n keda
```

b.

/var/audit-policy/ ディレクトリーに移動します。

```
sh-4.4$ cd /var/audit-policy/
```

c.

利用可能なログを一覧表示します。

```
sh-4.4$ ls
```

出力例

```
log-2023.02.17-14:50 policy.yaml
```


d.

必要に応じてログを表示します。

```
sh-4.4$ cat <log_name>/<pvc_name>|grep -i <log_level> ❶
```

❶

オプション: `grep` コマンドを使用して、表示するログレベル (Metadata、Request、RequestResponse) を指定できます。

以下に例を示します。

```
sh-4.4$ cat log-2023.02.17-14:50/pvc-audit-log|grep -i Request
```

出力例

```
...
{"kind":"Event","apiVersion":"audit.k8s.io/v1","level":"Request","auditID":"63e7f68c-04ec-4f4d-8749-bf1656572a41","stage":"ResponseComplete","requestURI":"/openapi/v2","verb":"get","user":{"username":"system:aggregator","groups":["system:authenticated"]},"sourceIPs":["10.128.0.1"],"responseStatus":{"metadata":{"code":304},"requestReceivedTimestamp":"2023-02-17T13:12:55.035478Z","stageTimestamp":"2023-02-17T13:12:55.038346Z","annotations":{"authorization.k8s.io/decision":"allow","authorization.k8s.io/reason":"RBAC: allowed by ClusterRoleBinding \"system:discovery\" of ClusterRole \"system:discovery\" to Group \"system:authenticated\""}}}
...
```

3.8. デバッグデータの収集

サポートケースを作成する際、ご使用のクラスタのデバッグ情報を Red Hat サポートに提供していただくと Red Hat のサポートに役立ちます。

問題のトラブルシューティングに使用するため、以下の情報を提供してください。

- **must-gather** ツールを使用して収集されるデータ。
- 一意のクラスター ID。

must-gather ツールを使用して、以下を含む **Custom Metrics Autoscaler Operator** とそのコンポーネントに関するデータを収集できます。

- **keda namespace** とその子オブジェクト。
- **Custom Metric Autoscaler Operator** のインストールオブジェクト。
- **Custom Metric Autoscaler Operator** の CRD オブジェクト。

3.8.1. デバッグデータの収集

以下のコマンドは、**Custom Metrics Autoscaler Operator** の **must-gather** ツールを実行します。

```
$ oc adm must-gather --image="$(oc get packagemanifests openshift-custom-metrics-autoscaler-operator \
-n openshift-marketplace \
-o jsonpath='{.status.channels[?(@.name=="stable")].currentCSVDesc.annotations.containerImage}')
```



注記

標準の Red Hat OpenShift Service on AWS の **must-gather** コマンドである **oc adm must-gather** は、**Custom Metrics Autoscaler Operator** のデータを収集しません。

前提条件

- Red Hat OpenShift Service on AWS に、**dedicated-admin** ロールを持つユーザーとしてログインしている。

- Red Hat OpenShift Service on AWS CLI (oc) がインストールされている。

手順

1. `must-gather` データを保存するディレクトリーに移動します。
2. 以下のいずれかを実行します。

- Custom Metrics Autoscaler Operator の `must-gather` データのみを取得するには、以下のコマンドを使用します。

```
$ oc adm must-gather --image="$(oc get packagemanifests openshift-custom-
metrics-autoscaler-operator \
-n openshift-marketplace \
-o jsonpath='{.status.channels[?
(@.name=="stable")].currentCSVDesc.annotations.containerImage}')"
```

`must-gather` コマンドのカスタムイメージは、Operator パッケージマニフェストから直接プルされます。そうすることで、Custom Metric Autoscaler Operator が利用可能なクラスター上で機能します。

- Custom Metric Autoscaler Operator 情報に加えてデフォルトの `must-gather` データを収集するには、以下を実行します。

- a. 以下のコマンドを使用して Custom Metrics Autoscaler Operator イメージを取得し、これを環境変数として設定します。

```
$ IMAGE="$(oc get packagemanifests openshift-custom-metrics-autoscaler-
operator \
-n openshift-marketplace \
-o jsonpath='{.status.channels[?
(@.name=="stable")].currentCSVDesc.annotations.containerImage}')"
```

- b. Custom Metrics Autoscaler Operator イメージで `oc adm must-gather` を使用するには、以下を実行します。

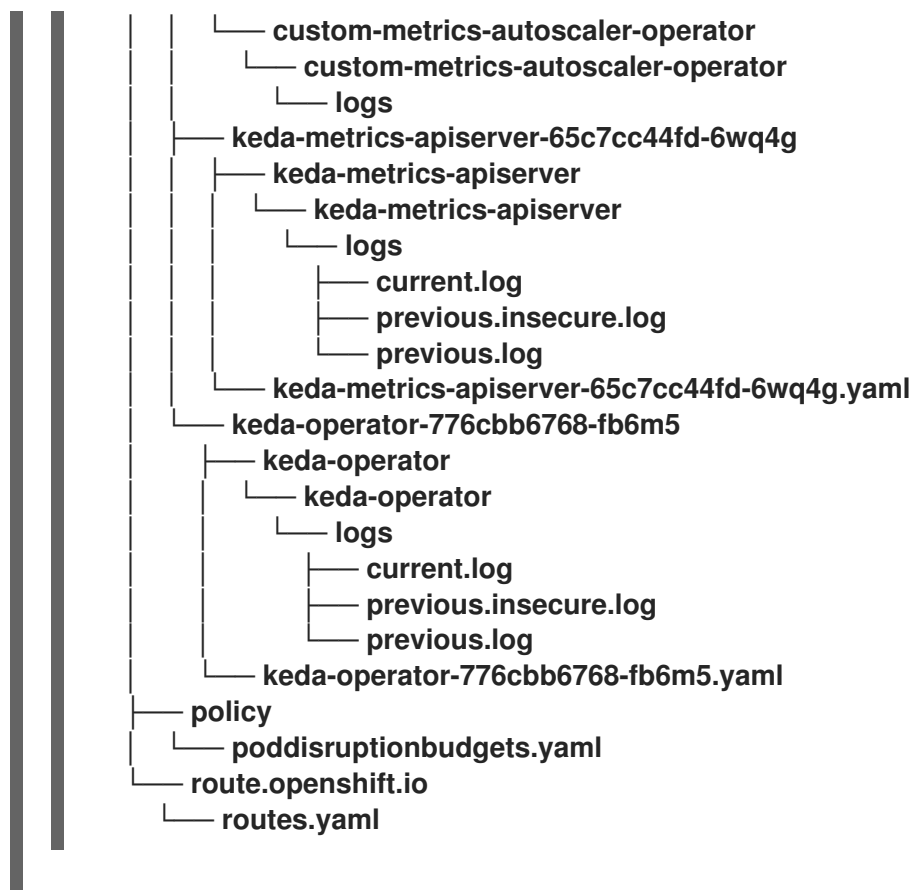
```
$ oc adm must-gather --image-stream=openshift/must-gather --
image=${IMAGE}
```

例3.1 Custom Metric Autoscaler の must-gather 出力例:

```

├── keda
│   ├── apps
│   │   ├── daemonsets.yaml
│   │   ├── deployments.yaml
│   │   ├── replicasetsets.yaml
│   │   └── statefulsets.yaml
│   ├── apps.openshift.io
│   │   └── deploymentconfigs.yaml
│   ├── autoscaling
│   │   └── horizontalpodautoscalers.yaml
│   ├── batch
│   │   ├── cronjobs.yaml
│   │   └── jobs.yaml
│   ├── build.openshift.io
│   │   ├── buildconfigs.yaml
│   │   └── builds.yaml
│   ├── core
│   │   ├── configmaps.yaml
│   │   ├── endpoints.yaml
│   │   ├── events.yaml
│   │   ├── persistentvolumeclaims.yaml
│   │   ├── pods.yaml
│   │   ├── replicationcontrollers.yaml
│   │   ├── secrets.yaml
│   │   └── services.yaml
│   ├── discovery.k8s.io
│   │   └── endpointslices.yaml
│   ├── image.openshift.io
│   │   └── imagestreams.yaml
│   ├── k8s.ovn.org
│   │   ├── egressfirewalls.yaml
│   │   └── egressqoses.yaml
│   ├── keda.sh
│   │   ├── kedacontrollers
│   │   │   └── keda.yaml
│   │   ├── scaledobjects
│   │   │   └── example-scaledobject.yaml
│   │   └── triggerauthentications
│   │       └── example-triggerauthentication.yaml
│   ├── monitoring.coreos.com
│   │   └── servicemonitors.yaml
│   ├── networking.k8s.io
│   │   └── networkpolicies.yaml
│   ├── keda.yaml
│   ├── pods
│   │   ├── custom-metrics-autoscaler-operator-58bd9f458-ptgwx
│   │   │   ├── custom-metrics-autoscaler-operator
│   │   │   │   └── custom-metrics-autoscaler-operator
│   │   │   │       └── logs
│   │   │   │           ├── current.log
│   │   │   │           ├── previous.insecure.log
│   │   │   │           └── previous.log
│   │   └── custom-metrics-autoscaler-operator-58bd9f458-ptgwx.yaml
│   └── custom-metrics-autoscaler-operator-58bd9f458-thbsh

```



3.

作業ディレクトリーに作成された **must-gather** ディレクトリーから圧縮ファイルを作成します。たとえば、Linux オペレーティングシステムを使用するコンピューターで以下のコマンドを実行します。

```
$ tar cvaf must-gather.tar.gz must-gather.local.5421342344627712289/ 1
```

1

must-gather-local.5421342344627712289/ を実際のディレクトリー名に置き換えます。

4.

圧縮ファイルを [Red Hat カスタマーポータル](#) で作成したサポートケースに添付します。

3.9. OPERATOR メトリクスの表示

Custom Metrics Autoscaler Operator は、クラスター上のモニタリングコンポーネントからプルした、すぐに使用可能なメトリクスを公開します。**Prometheus Query Language (PromQL)** を使用してメトリクスをクエリーし、問題を分析および診断できます。コントローラー Pod の再起動時にすべてのメトリクスがリセットされます。

3.9.1. パフォーマンスメトリクスへのアクセス

Red Hat OpenShift Service on AWS Web コンソールを使用し、メトリクスにアクセスしてクエリーを実行できます。

手順

1. Red Hat OpenShift Service on AWS Web コンソールで **Administrator** パースペクティブを選択します。
2. **Observe** → **Metrics** の順に選択します。
3. カスタムクエリーを作成するには、**PromQL** クエリーを **Expression** フィールドに追加します。
4. 複数のクエリーを追加するには、**Add Query** を選択します。

3.9.1.1. 提供される Operator メトリクス

Custom Metrics Autoscaler Operator は、以下のメトリクスを公開します。メトリクスは、Red Hat OpenShift Service on AWS Web コンソールを使用して表示できます。

表3.1 Custom Metric Autoscaler Operator メトリクス

| メトリクス名 | 説明 |
|-------------------------------------|---|
| keda_scaler_activity | 特定のスケーラーがアクティブか非アクティブかを示します。値が 1 の場合はスケーラーがアクティブであることを示し、値が 0 の場合はスケーラーが非アクティブであることを示します。 |
| keda_scaler_metrics_value | 各スケーラーのメトリクスの現在の値。ターゲットの平均を計算する際に Horizontal Pod Autoscaler (HPA) によって使用されます。 |
| keda_scaler_metrics_lateness | 各スケーラーから現在のメトリクスを取得する際のレイテンシー。 |
| keda_scaler_errors | 各スケーラーで発生したエラーの数。 |
| keda_scaler_errors_total | すべてのスケーラーで発生したエラーの合計数。 |
| keda_scaled_object_errors | スケーリングされた各オブジェクトで発生したエラーの数。 |

| メトリクス名 | 説明 |
|-----------------------------|---|
| keda_resource_totals | 各カスタムリソースタイプの各 namespace における Custom Metrics Autoscaler カスタムリソースの合計数。 |
| keda_trigger_totals | トリガータイプごとのトリガー合計数。 |

Custom Metrics Autoscaler Admission Webhook メトリクス

Custom Metrics Autoscaler Admission Webhook は、以下の Prometheus メトリクスも公開します。

| メトリクス名 | 説明 |
|---|----------------------|
| keda_scaled_object_validation_total | スケーリングされたオブジェクトの検証数。 |
| keda_scaled_object_validation_errors | 検証エラーの数。 |

3.10. カスタムメトリクスオートスケーラーの追加方法について

カスタムメトリクスオートスケーラーを追加するには、デプロイメント、ステートフルセット、またはカスタムリソース用の ScaledObject カスタムリソースを作成します。ジョブの ScaledJob カスタムリソースを作成します。

スケーリングするワークロードごとに、スケーリングされたオブジェクトを 1 つだけ作成できます。スケーリングされたオブジェクトと水平 Pod オートスケーラー (HPA) は、同じワークロードで使用できません。

3.10.1. ワークロードへのカスタムメトリクスオートスケーラーの追加

Deployment、StatefulSet、または custom resource オブジェクトによって作成されるワークロード用のカスタムメトリクスオートスケーラーを作成できます。

前提条件

- Custom Metrics Autoscaler Operator をインストールしている。
- CPU またはメモリーに基づくスケーリングにカスタムメトリクスオートスケーラーを使用

する場合:

- クラスター管理者は、クラスターメトリクスを適切に設定する必要があります。メトリクスが設定されているかどうかは、`oc describe PodMetrics <pod-name>` コマンドを使用して判断できます。メトリクスが設定されている場合、出力は以下の Usage の下にある CPU と Memory のように表示されます。

```
$ oc describe PodMetrics openshift-kube-scheduler-ip-10-0-135-131.ec2.internal
```

出力例

```
Name:      openshift-kube-scheduler-ip-10-0-135-131.ec2.internal
Namespace: openshift-kube-scheduler
Labels:    <none>
Annotations: <none>
API Version: metrics.k8s.io/v1beta1
Containers:
  Name: wait-for-host-port
  Usage:
    Memory: 0
  Name: scheduler
  Usage:
    Cpu: 8m
    Memory: 45440Ki
Kind: PodMetrics
Metadata:
  Creation Timestamp: 2019-05-23T18:47:56Z
  Self Link: /apis/metrics.k8s.io/v1beta1/namespaces/openshift-kube-scheduler/pods/openshift-kube-scheduler-ip-10-0-135-131.ec2.internal
  Timestamp: 2019-05-23T18:47:56Z
  Window: 1m0s
  Events: <none>
```

- スケーリングするオブジェクトに関連付けられた Pod には、指定されたメモリーと CPU の制限が含まれている必要があります。以下に例を示します。

Pod 仕様の例

```
apiVersion: v1
kind: Pod
# ...
```



```
spec:
  containers:
  - name: app
    image: images.my-company.example/app:v4
  resources:
    limits:
      memory: "128Mi"
      cpu: "500m"
# ...
```

手順

1.

以下のような YAML ファイルを作成します。名前 <2>、オブジェクト名 <4>、およびオブジェクトの種類 <5> のみが必要です。

スケーリングされたオブジェクトの例

```
apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  annotations:
    autoscaling.keda.sh/paused-replicas: "0" ①
  name: scaledobject ②
  namespace: my-namespace
spec:
  scaleTargetRef:
    apiVersion: apps/v1 ③
    name: example-deployment ④
    kind: Deployment ⑤
    envSourceContainerName: .spec.template.spec.containers[0] ⑥
  cooldownPeriod: 200 ⑦
  maxReplicaCount: 100 ⑧
  minReplicaCount: 0 ⑨
  metricsServer: ⑩
  auditConfig:
    logFormat: "json"
    logOutputVolumeClaim: "persistentVolumeClaimName"
    policy:
      rules:
      - level: Metadata
        omitStages: "RequestReceived"
        omitManagedFields: false
  lifetime:
    maxAge: "2"
    maxBackup: "1"
```

```
    maxSize: "50"
  fallback: 11
  failureThreshold: 3
  replicas: 6
  pollingInterval: 30 12
  advanced:
    restoreToOriginalReplicaCount: false 13
    horizontalPodAutoscalerConfig:
      name: keda-hpa-scale-down 14
      behavior: 15
      scaleDown:
        stabilizationWindowSeconds: 300
        policies:
          - type: Percent
            value: 100
            periodSeconds: 15
  triggers:
    - type: prometheus 16
      metadata:
        serverAddress: https://thanos-querier.openshift-monitoring.svc.cluster.local:9092
        namespace: kedatest
        metricName: http_requests_total
        threshold: '5'
        query: sum(rate(http_requests_total{job="test-app"}[1m]))
        authModes: basic
      authenticationRef: 17
        name: prom-triggerauthentication
        kind: TriggerAuthentication
```

1

オプション: 「ワークロードのカスタムメトリクスオートスケーラーの一時停止」セクションで説明されているように、Custom Metrics Autoscaler Operator がレプリカを指定された値にスケールし、自動スケールリングを停止するよう指定します。

2

このカスタムメトリクスオートスケーラーの名前を指定します。

3

オプション: ターゲットリソースの API バージョンを指定します。デフォルトは apps/v1 です。

4

スケールリングするオブジェクトの名前を指定します。

5

6

オプション: カスタムメトリクスオートスケーラーがシークレットなどを保持する環境変数を取得する、ターゲットリソース内のコンテナの名前を指定します。デフォルトは `.spec.template.spec.containers[0]` です。

7

オプション: `minReplicaCount` が 0 に設定されている場合、最後のトリガーが報告されてからデプロイメントを 0 にスケールバックするまでの待機時間を秒単位で指定します。デフォルトは 300 です。

8

オプション: スケールアップ時のレプリカの最大数を指定します。デフォルトは 100 です。

9

オプション: スケールダウン時のレプリカの最小数を指定します。

10

オプション: 「監査ログの設定」セクションで説明されているように、監査ログのパラメーターを指定します。

11

オプション: `failureThreshold` パラメーターで定義された回数だけスケーラーがソースからメトリクスを取得できなかった場合に、フォールバックするレプリカの数を指定します。フォールバック動作の詳細は、[KEDA のドキュメント](#) を参照してください。

12

オプション: 各トリガーをチェックする間隔を秒単位で指定します。デフォルトは 30 です。

13

オプション: スケーリングされたオブジェクトが削除された後に、ターゲットリソースを元のレプリカ数にスケールバックするかどうかを指定します。デフォルトは `false` で、スケールアップされたオブジェクトが削除されたときのレプリカ数をそのまま保持します。

14

15

オプション: 「スケーリングポリシー」セクションで説明されているように、Pod をスケールアップまたはスケールダウンするレートを制御するために使用するスケーリングポリシーを指定します。

16

「カスタムメトリクスオートスケーラートリガーについて」セクションで説明されているように、スケーリングの基準として使用するトリガーを指定します。この例では、Red Hat OpenShift Service on AWS モニタリングを使用します。

17

オプション: トリガー認証またはクラスタートリガー認証を指定します。詳細は、関連情報 セクションの カスタムメトリクスオートスケーラー認証について を参照してください。

- トリガー認証を使用するには、`TriggerAuthentication` と入力します。これはデフォルトになります。
- クラスタートリガー認証を使用するには、`ClusterTriggerAuthentication` と入力します。

2.

次のコマンドを実行して、カスタムメトリクスオートスケーラーを作成します。

```
$ oc create -f <filename>.yaml
```

検証

- コマンド出力を表示して、カスタムメトリクスオートスケーラーが作成されたことを確認します。

```
$ oc get scaledobject <scaled_object_name>
```

出力例

```
NAME          SCALETARGETKIND  SCALETARGETNAME  MIN  MAX  TRIGGERS
AUTHENTICATION  READY  ACTIVE  FALLBACK  AGE
```

```
scaledobject apps/v1.Deployment example-deployment 0 50 prometheus
prom-triggerauthentication True True True 17s
```

出力の次のフィールドに注意してください。

- **TRIGGERS:** 使用されているトリガーまたはスケーラーを示します。
- **AUTHENTICATION:** 使用されているトリガー認証の名前を示します。
- **READY:** スケーリングされたオブジェクトがスケーリングを開始する準備ができているかどうかを示します。
 - **True** の場合、スケーリングされたオブジェクトの準備は完了しています。
 - **False** の場合、作成したオブジェクトの1つ以上に問題があるため、スケーリングされたオブジェクトの準備は完了していません。
- **ACTIVE:** スケーリングが行われているかどうかを示します。
 - **True** の場合、スケーリングが行われています。
 - **False** の場合、メトリクスがないか、作成したオブジェクトの1つ以上に問題があるため、スケーリングは行われていません。
- **FALLBACK:** カスタムメトリクスオートスケーラーがソースからメトリクスを取得できるかどうかを示します。
 - **False** の場合、カスタムメトリクスオートスケーラーはメトリクスを取得していません。
 -

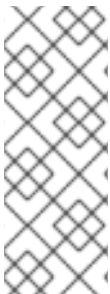
True の場合、メトリクスがないか、作成したオブジェクトの 1 つ以上に問題があるため、カスタムメトリクスオートスケーラーはメトリクスを取得しています。

3.10.2. 関連情報

- [カスタムメトリクスオートスケーラートリガー認証について](#)

3.11. CUSTOM METRICS AUTOSCALER OPERATOR の削除

Red Hat OpenShift Service on AWS クラスターからカスタムメトリクスオートスケーラーを削除できます。Custom Metrics Autoscaler Operator を削除した後、潜在的な問題を回避するために、Operator に関連付けられている他のコンポーネントを削除します。



注記

最初に KedaController カスタムリソース (CR) を削除します。KedaController CR を削除しないと、keda プロジェクトを削除するときに Red Hat OpenShift Service on AWS がハングする可能性があります。CR を削除する前に Custom Metrics Autoscaler Operator を削除すると、CR を削除することはできません。

3.11.1. Custom Metrics Autoscaler Operator のアンインストール


以下の手順を使用して、Red Hat OpenShift Service on AWS クラスターからカスタムメトリクスオートスケーラーを削除します。

前提条件

- Custom Metrics Autoscaler Operator をインストールしている。

手順

1. Red Hat OpenShift Service on AWS Web コンソールで、Operators → Installed Operators をクリックします。
2. keda プロジェクトに切り替えます。
3. KedaController カスタムリソースを削除します。

- a. **CustomMetricsAutoscaler Operator** を見つけて、**KedaController** タブをクリックします。
 - b. カスタムリソースを見つけてから、**Delete KedaController** をクリックします。
 - c. **Uninstall** をクリックします。
4. **Custom Metrics Autoscaler Operator** を削除します。
- a. **Operators** → **Installed Operators** をクリックします。
 - b. **CustomMetricsAutoscaler Operator** を見つけて **Options** メニュー

をクリックし、**Uninstall Operator** を選択します。
 - c. **Uninstall** をクリックします。
5. オプション: OpenShift CLI を使用して、カスタムメトリクスオートスケーラーのコンポーネントを削除します。
- a. カスタムメトリクスオートスケーラーの CRD を削除します。
 - **clustertriggerauthentications.keda.sh**
 - **kedacontrollers.keda.sh**
 - **scaledjobs.keda.sh**
 - **scaledobjects.keda.sh**

- **triggerauthentications.keda.sh**

```
$ oc delete crd clustertriggerauthentications.keda.sh kedacontrollers.keda.sh  
scaledjobs.keda.sh scaledobjects.keda.sh triggerauthentications.keda.sh
```

CRD を削除すると、関連付けられたロール、クラスターロール、およびロールバインディングが削除されます。ただし、手動で削除する必要のあるクラスターロールがいくつかあります。

- b. カスタムメトリクスオートスケーラークラスターのロールをリスト表示します。

```
$ oc get clusterrole | grep keda.sh
```

- c. リスト表示されているカスタムメトリクスオートスケーラークラスターのロールを削除します。以下に例を示します。

```
$ oc delete clusterrole.keda.sh-v1alpha1-admin
```

- d. カスタムメトリクスオートスケーラークラスターのロールバインディングをリスト表示します。

```
$ oc get clusterrolebinding | grep keda.sh
```

- e. リスト表示されているカスタムメトリクスオートスケーラークラスターのロールバインディングを削除します。以下に例を示します。

```
$ oc delete clusterrolebinding.keda.sh-v1alpha1-admin
```

6. カスタムメトリクスオートスケーラーのプロジェクトを削除します。

```
$ oc delete project keda
```

7. **Cluster Metric Autoscaler Operator** を削除します。

```
$ oc delete operator/openshift-custom-metrics-autoscaler-operator.keda
```


第4章 POD のノードへの配置の制御 (スケジューリング)

4.1. スケジューラーによる POD 配置の制御

Pod のスケジューリングは、クラスター内のノードへの新規 Pod の配置を決定する内部プロセスです。

スケジューラーコードは、新規 Pod の作成時にそれらを確認し、それらをホストするのに最も適したノードを識別します。次に、マスター API を使用して Pod のバインディング (Pod とノードのバインディング) を作成します。

デフォルトの Pod スケジューリング

Red Hat OpenShift Service on AWS には、ほとんどのユーザーのニーズに対応するデフォルトのスケジューラーが付属しています。デフォルトスケジューラーは、Pod に最適なノードを判別するために固有のツールとカスタマイズ可能なツールの両方を使用します。

詳細な Pod スケジューリング

新しい Pod の配置場所をより詳細に制御する必要がある場合、Red Hat OpenShift Service on AWS の詳細スケジューリング機能を使用すると、必ずまたは可能な限り、特定ノード上または特定の Pod とともに実行するよう Pod を設定できます。

以下のスケジューリング機能を使用して、Pod の配置を制御できます。

- [Pod のアフィニティーおよび非アフィニティールール](#)
- [ノードのアフィニティー](#)
- [ノードセレクター](#)
- [テイントおよび容認 \(Toleration\)](#)
- [ノードのオーバーコミット](#)

4.1.1. デフォルトスケジューラーについて

Red Hat OpenShift Service on AWS のデフォルトの Pod スケジューラーは、クラスター内のノードへの新しい Pod の配置を決定します。スケジューラーは Pod からのデータを読み取り、設定されるプロファイルに基づいて適切なノードを見つけます。これは完全に独立した機能であり、スタンドアロンソリューションです。Pod を変更することはなく、Pod を特定ノードに関連付ける Pod のバインディングを作成します。

4.1.1.1. デフォルトスケジューリングについて

既存の汎用スケジューラーはプラットフォームで提供されるデフォルトのスケジューラー エンジンであり、Pod をホストするノードを 3 つの手順で選択します。

ノードのフィルター

利用可能なノードは、指定される制約や要件に基づいてフィルターされます。フィルターは、各ノードで述語またはフィルターというフィルター関数の一覧を使用して実行されます。

フィルターされたノードリストの優先順位付け

優先順位付けは、各ノードに一連の優先度 または スコアリング 関数を実行することによって行われます。この関数は 0 -10 までのスコアをノードに割り当て、0 は不適切であることを示し、10 は Pod のホストに適していることを示します。スケジューラー設定は、それぞれのスコアリング関数について単純な重み (正の数値) を取ることができます。各スコアリング関数で指定されるノードのスコアは重み (ほとんどのスコアのデフォルトの重みは 1) で乗算され、すべてのスコアで指定されるそれぞれのノードのスコアを追加して組み合わせられます。この重み属性は、一部のスコアにより重きを置くようにするなどの目的で管理者によって使用されます。

最適ノードの選択

ノードの並び替えはそれらのスコアに基づいて行われ、最高のスコアを持つノードが Pod をホストするように選択されます。複数のノードに同じ高スコアが付けられている場合、それらのいずれかがランダムに選択されます。

4.1.2. スケジューラーの使用例

Red Hat OpenShift Service on AWS 内でのスケジューリングの重要な使用例として、柔軟なアフィニティーと非アフィニティーポリシーのサポートを挙げることができます。

4.1.2.1. アフィニティー

管理者は、任意のトポロジーレベルまたは複数のレベルでもアフィニティーを指定できるようにスケジューラーを設定することができます。特定レベルのアフィニティーは、同じサービスに属するすべての Pod が同じレベルに属するノードにスケジュールされることを示します。これは、管理者がピア Pod が地理的に離れ過ぎないようにすることでアプリケーションの待機時間の要件に対応します。同じ

アフィニティグループ内で Pod をホストするために利用できるノードがない場合、Pod はスケジュールされません。

Pod がスケジュールされる場所をより詳細に制御する必要がある場合は、[ノードアフィニティールールを使用したノードでの Pod 配置の制御](#) および [アフィニティールールと非アフィニティールールを使用した他の Pod に対する相対的なPod の配置](#) を参照してください。

これらの高度なスケジューリング機能を使うと、管理者は Pod をスケジューリングするノードを指定でき、他の Pod との比較でスケジューリングを実行したり、拒否したりすることができます。

4.1.2.2. 非アフィニティ

管理者は、任意のトポロジーレベルまたは複数のレベルでも非アフィニティを設定できるようにスケジューラーを設定することができます。特定レベルの非アフィニティ (または分散) は、同じサービスに属するすべての Pod が該当レベルに属するノード全体に分散されることを示します。これにより、アプリケーションが高可用性の目的で適正に分散されます。スケジューラーは、可能な限り均等になるようにすべての適用可能なノード全体にサービス Pod を配置しようとします。

Pod がスケジュールされる場所をより詳細に制御する必要がある場合は、[ノードアフィニティールールを使用したノードでの Pod 配置の制御](#) および [アフィニティールールと非アフィニティールールを使用した他の Pod に対する相対的なPod の配置](#) を参照してください。

これらの高度なスケジューリング機能を使うと、管理者は Pod をスケジューリングするノードを指定でき、他の Pod との比較でスケジューリングを実行したり、拒否したりすることができます。

4.2. アフィニティールールと非アフィニティールールの使用による他の POD との相対での POD の配置

アフィニティとは、スケジューリングするノードを制御する Pod の特性です。非アフィニティとは、Pod がスケジューリングされることを拒否する Pod の特性です。

Red Hat OpenShift Service on AWS では、Pod のアフィニティ と Pod の非アフィニティ によって、他の Pod のキー/値ラベルに基づいて、Pod のスケジューリングに適したノードを制限できます。

4.2.1. Pod のアフィニティについて

Pod のアフィニティ と Pod の非アフィニティ によって、他の Pod のキー/値ラベルに基づいて、Pod をスケジューリングすることに適したノードを制限することができます。

- Pod のアフィニティーはスケジューラーに対し、新規 Pod のラベルセクターが現在の Pod のラベルに一致する場合に他の Pod と同じノードで新規 Pod を見つけるように指示します。
- Pod の非アフィニティーは、新規 Pod のラベルセクターが現在の Pod のラベルに一致する場合に、同じラベルを持つ Pod と同じノードで新規 Pod を見つけることを禁止します。

たとえば、アフィニティールールを使用することで、サービス内で、または他のサービスの Pod との関連で Pod を分散したり、パックしたりすることができます。非アフィニティールールにより、特定のサービスの Pod がそのサービスの Pod のパフォーマンスに干渉すると見なされる別のサービスの Pod と同じノードでスケジュールされることを防ぐことができます。または、関連する障害を減らすために複数のノード、アベイラビリティゾーン、またはアベイラビリティセットの間でサービスの Pod を分散することもできます。



注記

ラベルセクターは、複数の Pod デプロイメントを持つ Pod に一致する可能性があります。非アフィニティールールを設定して Pod が一致しないようにする場合は、一意のラベル組み合わせを使用します。

Pod のアフィニティーには、**required (必須)** および **preferred (優先)** の 2 つのタイプがあります。

Pod をノードにスケジュールする前に、**required (必須)** ルールを満たしている必要があります。**preferred (優先)** ルールは、ルールを満たす場合に、スケジューラーはルールの実施を試行しますが、その実施が必ずしも保証される訳ではありません。



注記

Pod の優先順位およびプリエンプレションの設定により、スケジューラーはアフィニティーの要件に違反しなければ Pod の適切なノードを見つけれない可能性があります。その場合、Pod はスケジュールされない可能性があります。

この状態を防ぐには、優先順位が等しい Pod との Pod のアフィニティーの設定を慎重に行ってください。

Pod のアフィニティー/非アフィニティーは Pod 仕様ファイルで設定します。**required (必須)** ルー

ル、preferred (優先) ルールのいずれか、その両方を指定することができます。両方を指定する場合、ノードは最初に required (必須) ルールを満たす必要があり、その後に preferred (優先) ルールを満たそうとします。

以下の例は、Pod のアフィニティーおよび非アフィニティーに設定される Pod 仕様を示しています。

この例では、Pod のアフィニティールールは ノードにキー security と値 S1 を持つラベルの付いた 1 つ以上の Pod がすでに行われている場合にのみ Pod をノードにスケジューリングできることを示しています。Pod の非アフィニティールールは、ノードがキー security と値 S2 を持つラベルが付いた Pod がすでに行われている場合は Pod をノードにスケジューリングしないように設定することを示しています。

Pod のアフィニティーが設定された Pod 設定ファイルのサンプル

```

apiVersion: v1
kind: Pod
metadata:
  name: with-pod-affinity
spec:
  securityContext:
    runAsNonRoot: true
  seccompProfile:
    type: RuntimeDefault
  affinity:
    podAffinity: ❶
      requiredDuringSchedulingIgnoredDuringExecution: ❷
        - labelSelector:
            matchExpressions:
              - key: security ❸
                operator: In ❹
                values:
                  - S1 ❺
            topologyKey: topology.kubernetes.io/zone
  containers:
    - name: with-pod-affinity
      image: docker.io/ocpqe/hello-pod
      securityContext:
        allowPrivilegeEscalation: false
      capabilities:
        drop: [ALL]

```

Pod のアフィニティーを設定するためのスタンザです。

2

required (必須) ルールを定義します。

3

5

ルールを適用するために一致している必要のあるキーと値 (ラベル) です。

4

演算子は、既存 Pod のラベルと新規 Pod の仕様の `matchExpression` パラメーターの値のセットの間の関係を表します。これには `In`、`NotIn`、`Exists`、または `DoesNotExist` のいずれかを使用できます。

Pod の非アフィニティーが設定された Pod 設定ファイルのサンプル

```

apiVersion: v1
kind: Pod
metadata:
  name: with-pod-antiaffinity
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  affinity:
    podAntiAffinity: 1
      preferredDuringSchedulingIgnoredDuringExecution: 2
      - weight: 100 3
        podAffinityTerm:
          labelSelector:
            matchExpressions:
              - key: security 4
                operator: In 5
                values:
                  - S2
            topologyKey: kubernetes.io/hostname
  containers:
    - name: with-pod-affinity
      image: docker.io/ocpqe/hello-pod
      securityContext:
        allowPrivilegeEscalation: false
      capabilities:
        drop: [ALL]

```

1

Pod の非アフィニティーを設定するためのスタンザです。

2

preferred (優先) ルールを定義します。

3

preferred (優先) ルールの重みを指定します。最も高い重みを持つノードが優先されます。

4

非アフィニティールールが適用される時を決定する Pod ラベルの説明です。ラベルのキーおよび値を指定します。

5

演算子は、既存 Pod のラベルと新規 Pod の仕様の matchExpression パラメーターの値のセットの関係を表します。これには In、NotIn、Exists、または DoesNotExist のいずれかを使用できます。

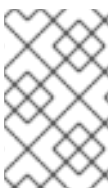


注記

ノードのラベルに、Pod のノードのアフィニティールールを満たさなくなるような結果になる変更がランタイム時に生じる場合も、Pod はノードで引き続き実行されません。

4.2.2. Pod アフィニティールールの設定

以下の手順は、ラベルの付いた Pod と Pod のスケジュールを可能にするアフィニティーを使用する Pod を作成する 2 つの Pod の単純な設定を示しています。



注記

アフィニティーをスケジュールされた Pod に直接追加することはできません。

手順

1. Pod 仕様の特定のラベルの付いた Pod を作成します。

- a. 以下の内容を含む YAML ファイルを作成します。

```
apiVersion: v1
kind: Pod
metadata:
  name: security-s1
  labels:
    security: S1
spec:
  securityContext:
    runAsNonRoot: true
  seccompProfile:
    type: RuntimeDefault
  containers:
  - name: security-s1
    image: docker.io/ocpqe/hello-pod
    securityContext:
      runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
```

- b. Pod を作成します。

```
$ oc create -f <pod-spec>.yaml
```

2. 他の Pod の作成時に、以下のパラメーターを設定してアフィニティーを追加します。

- a. 以下の内容を含む YAML ファイルを作成します。

```
apiVersion: v1
kind: Pod
metadata:
  name: security-s1-east
  # ...
spec:
  affinity: ❶
  podAffinity:
    requiredDuringSchedulingIgnoredDuringExecution: ❷
  - labelSelector:
      matchExpressions:
        - key: security ❸
          values:
            - S1
```



```
operator: In 4
topologyKey: topology.kubernetes.io/zone 5
# ...
```

1

Pod のアフィニティーを追加します。

2

`requiredDuringSchedulingIgnoredDuringExecution` パラメーターまたは `preferredDuringSchedulingIgnoredDuringExecution` パラメーターを設定します。

3

満たす必要のある `key` および `values` を指定します。新規 Pod を他の Pod と共にスケジューリングする必要がある場合、最初の Pod のラベルと同じ `key` および `values` パラメーターを使用します。

4

Operator を指定します。演算子は `In`、`NotIn`、`Exists`、または `DoesNotExist` にすることができます。たとえば、演算子 `In` を使用してラベルをノードで必要になるようにします。

5

`topologyKey` を指定します。これは、システムがトポロジードメインを表すために使用する事前にデータが設定された **Kubernetes ラベル** です。

b.

Pod を作成します。

```
$ oc create -f <pod-spec>.yaml
```

4.2.3. Pod 非アフィニティールールの設定

以下の手順は、ラベルの付いた Pod と Pod のスケジューリングの禁止を試行する非アフィニティーの `preferred` (優先) ルールを使用する Pod を作成する 2 つの Pod の単純な設定を示しています。



注記

アフィニティーをスケジューリングされた Pod に直接追加することはできません。

手順

1. Pod 仕様の特定のラベルの付いた Pod を作成します。

- a. 以下の内容を含む YAML ファイルを作成します。

```
apiVersion: v1
kind: Pod
metadata:
  name: security-s1
  labels:
    security: S1
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
  - name: security-s1
    image: docker.io/ocpqe/hello-pod
    securityContext:
      allowPrivilegeEscalation: false
    capabilities:
      drop: [ALL]
```

- b. Pod を作成します。

```
$ oc create -f <pod-spec>.yaml
```

2. 他の Pod の作成時に、以下のパラメーターを設定します。

- a. 以下の内容を含む YAML ファイルを作成します。

```
apiVersion: v1
kind: Pod
metadata:
  name: security-s2-east
  # ...
spec:
  # ...
  affinity: ❶
  podAntiAffinity:
    preferredDuringSchedulingIgnoredDuringExecution: ❷
  - weight: 100 ❸
  podAffinityTerm:
```

```

labelSelector:
  matchExpressions:
    - key: security ④
      values:
        - S1
      operator: In ⑤
  topologyKey: kubernetes.io/hostname ⑥
# ...

```

①

Pod の非アフィニティーを追加します。

②

`requiredDuringSchedulingIgnoredDuringExecution` パラメーターまたは `preferredDuringSchedulingIgnoredDuringExecution` パラメーターを設定します。

③

優先ルールの場合、ノードの重みを 1 ~ 100 で指定します。最も高い重みを持つノードが優先されます。

④

満たす必要のある `key` および `values` を指定します。新規 Pod を他の Pod と共にスケジュールされないようにする必要がある場合、最初の Pod のラベルと同じ `key` および `values` パラメーターを使用します。

⑤

Operator を指定します。演算子は `In`、`NotIn`、`Exists`、または `DoesNotExist` にすることができます。たとえば、演算子 `In` を使用してラベルをノードで必要になるようにします。

⑥

`topologyKey` を指定します。これは、システムがトポロジードメインを表すために使用する事前にデータが設定された [Kubernetes ラベル](#) です。

b.

Pod を作成します。

```
$ oc create -f <pod-spec>.yaml
```

4.2.4. Pod のアフィニティールールと非アフィニティールール例

以下の例は、Pod のアフィニティーおよび非アフィニティーについて示しています。

4.2.4.1. Pod のアフィニティー

以下の例は、一致するラベルとラベルセレクターを持つ Pod についての Pod のアフィニティーを示しています。

- Pod team4 にはラベル team:4 が付けられています。

```
apiVersion: v1
kind: Pod
metadata:
  name: team4
  labels:
    team: "4"
# ...
spec:
  securityContext:
    runAsNonRoot: true
  seccompProfile:
    type: RuntimeDefault
  containers:
  - name: ocp
    image: docker.io/ocpqe/hello-pod
    securityContext:
      allowPrivilegeEscalation: false
    capabilities:
      drop: [ALL]
# ...
```

- Pod team4a には、podAffinity の下にラベルセレクター team:4 が付けられています。

```
apiVersion: v1
kind: Pod
metadata:
  name: team4a
# ...
spec:
  securityContext:
    runAsNonRoot: true
  seccompProfile:
    type: RuntimeDefault
  affinity:
    podAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
      - labelSelector:
          matchExpressions:
```

```

- key: team
  operator: In
  values:
  - "4"
  topologyKey: kubernetes.io/hostname
containers:
- name: pod-affinity
  image: docker.io/ocpqe/hello-pod
  securityContext:
    allowPrivilegeEscalation: false
  capabilities:
    drop: [ALL]
# ...

```

- team4a Pod は team4 Pod と同じノードにスケジュールされます。

4.2.4.2. Pod の非アフィニティー

以下の例は、一致するラベルとラベルセレクターを持つ Pod についての Pod の非アフィニティーを示しています。

- Pod pod-s1 にはラベル security:s1 が付けられています。

```

apiVersion: v1
kind: Pod
metadata:
  name: pod-s1
  labels:
    security: s1
# ...
spec:
  securityContext:
    runAsNonRoot: true
  seccompProfile:
    type: RuntimeDefault
containers:
- name: ocp
  image: docker.io/ocpqe/hello-pod
  securityContext:
    allowPrivilegeEscalation: false
  capabilities:
    drop: [ALL]
# ...

```

- Pod pod-s2 には、podAntiAffinity の下にラベルセレクター security:s1 が付けられています。

```

apiVersion: v1
kind: Pod
metadata:
  name: pod-s2
# ...
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  affinity:
    podAntiAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
      - labelSelector:
          matchExpressions:
            - key: security
              operator: In
              values:
                - s1
        topologyKey: kubernetes.io/hostname
  containers:
    - name: pod-antiaffinity
      image: docker.io/ocpqe/hello-pod
      securityContext:
        allowPrivilegeEscalation: false
      capabilities:
        drop: [ALL]
# ...

```

- Pod pod-s2 は pod-s1 と同じノードにスケジュールできません。

4.2.4.3. 一致するラベルのない Pod のアフィニティー

以下の例は、一致するラベルとラベルセレクターのない Pod についての Pod のアフィニティーを示しています。

- Pod pod-s1 にはラベル security:s1 が付けられています。

```

apiVersion: v1
kind: Pod
metadata:
  name: pod-s1
  labels:
    security: s1
# ...
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:

```

```

    type: RuntimeDefault
  containers:
  - name: ocp
    image: docker.io/ocpqe/hello-pod
    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
        drop: [ALL]
# ...

```

- Pod pod-s2 にはラベルセレクター security:s2 があります。

```

apiVersion: v1
kind: Pod
metadata:
  name: pod-s2
# ...
spec:
  securityContext:
    runAsNonRoot: true
  seccompProfile:
    type: RuntimeDefault
  affinity:
    podAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
      - labelSelector:
          matchExpressions:
          - key: security
            operator: In
            values:
            - s2
        topologyKey: kubernetes.io/hostname
  containers:
  - name: pod-affinity
    image: docker.io/ocpqe/hello-pod
    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
        drop: [ALL]
# ...

```

- Pod pod-s2 は、security:s2 ラベルの付いた Pod を持つノードがない場合はスケジュールされません。そのラベルの付いた他の Pod がない場合、新規 Pod は保留状態のままになります。

出力例

```

NAME     READY   STATUS    RESTARTS   AGE     IP          NODE
pod-s2   0/1     Pending  0           32s    <none>

```



4.3. ノードのアフィニティールールを使用したノード上での POD 配置の制御

アフィニティとは、スケジュールするノードを制御する Pod の特性です。

Red Hat OpenShift Service on AWS では、ノードのアフィニティとはスケジューラーが Pod を配置する場所を決定するために使用する一連のルールのことです。このルールは、ノードのカスタムラベルと Pod で指定されたラベルセレクターを使用して定義されます。

4.3.1. ノードのアフィニティについて

ノードのアフィニティにより、Pod がその配置に使用できるノードのグループに対してアフィニティを指定できます。ノード自体は配置に対して制御を行いません。

たとえば、Pod を特定の CPU を搭載したノードまたは特定のアベイラビリティゾーンにあるノードでのみ実行されるよう設定することができます。

ノードのアフィニティールールには、**required** (必須) および **preferred** (優先) の 2 つのタイプがあります。

Pod をノードにスケジュールする前に、**required** (必須) ルールを満たしている必要があります。**preferred** (優先) ルールは、ルールを満たす場合に、スケジューラーはルールの実施を試行しますが、その実施が必ずしも保証される訳ではありません。



注記

ランタイム時にノードのラベルに変更が生じ、その変更により Pod でのノードのアフィニティールールを満たさなくなる状態が生じるでも、Pod はノードで引き続き実行されます。

ノードのアフィニティは Pod 仕様ファイルで設定します。**required** (必須) ルール、**preferred** (優先) ルールのいずれか、その両方を指定することができます。両方を指定する場合、ノードは最初に **required** (必須) ルールを満たす必要があり、その後に **preferred** (優先) ルールを満たそうとします。

以下の例は、Pod をキーが `e2e-az-NorthSouth` で、その値が `e2e-az-North` または `e2e-az-South` のいずれかであるラベルの付いたノードに Pod を配置することを求めるルールが設定された Pod 仕様です。

ノードのアフィニティーの `required` (必須) ルールが設定された Pod 設定ファイルのサンプル

```

apiVersion: v1
kind: Pod
metadata:
  name: with-node-affinity
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  affinity:
    nodeAffinity: ①
      requiredDuringSchedulingIgnoredDuringExecution: ②
        nodeSelectorTerms:
          - matchExpressions:
              - key: e2e-az-NorthSouth ③
                operator: In ④
                values:
                  - e2e-az-North ⑤
                  - e2e-az-South ⑥
    containers:
      - name: with-node-affinity
        image: docker.io/ocpqe/hello-pod
        securityContext:
          allowPrivilegeEscalation: false
        capabilities:
          drop: [ALL]
# ...

```

①

ノードのアフィニティーを設定するためのスタンザです。

②

`required` (必須) ルールを定義します。

③ ⑤ ⑥

ルールを適用するために一致している必要のあるキー/値のペア (ラベル) です。

4

演算子は、ノードのラベルと Pod 仕様の `matchExpression` パラメーターの値のセットの間
の関係を表します。この値は、`In`、`NotIn`、`Exists`、または `DoesNotExist`、`Lt`、または `Gt` にする
ことができます。

以下の例は、キーが `e2e-az-EastWest` で、その値が `e2e-az-East` または `e2e-az-West` のラベルが付
いたノードに Pod を配置すること優先する `preferred` (優先) ルールが設定されたノード仕様です。

ノードのアフィニティーの `preferred` (優先) ルールが設定された Pod 設定ファイルのサンプル

```

apiVersion: v1
kind: Pod
metadata:
  name: with-node-affinity
spec:
  securityContext:
    runAsNonRoot: true
  seccompProfile:
    type: RuntimeDefault
  affinity:
    nodeAffinity: ①
      preferredDuringSchedulingIgnoredDuringExecution: ②
        - weight: 1 ③
          preference:
            matchExpressions:
              - key: e2e-az-EastWest ④
                operator: In ⑤
                values:
                  - e2e-az-East ⑥
                  - e2e-az-West ⑦
    containers:
      - name: with-node-affinity
        image: docker.io/ocpqe/hello-pod
        securityContext:
          allowPrivilegeEscalation: false
        capabilities:
          drop: [ALL]
# ...

```

1

ノードのアフィニティーを設定するためのスタンザです。

2

`preferred` (優先) ルールを定義します。

3

`preferred` (優先) ルールの重みを指定します。最も高い重みを持つノードが優先されます。

4 6 7

ルールを適用するために一致している必要のあるキー/値のペア (ラベル) です。

5

演算子は、ノードのラベルと Pod 仕様の `matchExpression` パラメーターの値のセットの間
の関係を表します。この値は、`In`、`NotIn`、`Exists`、または `DoesNotExist`、`Lt`、または `Gt` にする
ことができます。

ノードの非アフィニティー についての明示的な概念はありませんが、`NotIn` または `DoesNotExist`
演算子を使用すると、動作が複製されます。

注記

同じ Pod 設定でノードのアフィニティーとノードのセレクターを使用している場合
は、以下に注意してください。

- `nodeSelector` と `nodeAffinity` の両方を設定する場合、Pod が候補ノード
でスケジュールされるにはどちらの条件も満たしている必要があります。
- `nodeAffinity` タイプに関連付けられた複数の `nodeSelectorTerms` を指定
する場合、`nodeSelectorTerms` のいずれかが満たされている場合に Pod をノード
にスケジュールすることができます。
- `nodeSelectorTerms` に関連付けられた複数の `matchExpressions` を指定
する場合、すべての `matchExpressions` が満たされている場合にのみ Pod を
ノードにスケジュールすることができます。

4.3.2. ノードアフィニティーの required (必須) ルールの設定

Pod をノードにスケジュールする前に、required (必須) ルールを 満たしている必要があります。

手順

以下の手順は、ノードとスケジューラーがノードに配置する必要のある Pod を作成する単純な設定を示しています。

1. Pod 仕様の特定のラベルの付いた Pod を作成します。
 - a. 以下の内容を含む YAML ファイルを作成します。



注記

アフィニティーをスケジュールされた Pod に直接追加することはできません。

出力例

```

apiVersion: v1
kind: Pod
metadata:
  name: s1
spec:
  affinity: ①
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution: ②
        nodeSelectorTerms:
          - matchExpressions:
              - key: e2e-az-name ③
                values:
                  - e2e-az1
                  - e2e-az2
                operator: In ④
#...
```

①

2

`requiredDuringSchedulingIgnoredDuringExecution` パラメーターを設定します。

3

満たす必要のある `key` および `values` を指定します。新規 Pod を編集したノードにスケジュールする必要がある場合、ノードのラベルと同じ `key` および `values` パラメーターを使用します。

4

`Operator` を指定します。演算子は `In`、`NotIn`、`Exists`、または `DoesNotExist` にすることができます。たとえば、演算子 `In` を使用してラベルをノードで必要になるようにします。

b.

Pod を作成します。

```
$ oc create -f <file-name>.yaml
```

4.3.3. ノードアフィニティーの `preferred` (優先) ルールの設定

`preferred` (優先) ルールは、ルールを満たす場合に、スケジューラーはルールの実施を試行しますが、その実施が必ずしも保証される訳ではありません。

手順

以下の手順は、ノードとスケジューラーがノードに配置しようとする Pod を作成する単純な設定を示しています。

1.

特定のラベルの付いた Pod を作成します。

a.

以下の内容を含む YAML ファイルを作成します。



注記

アフィニティーをスケジュールされた Pod に直接追加することはできません。

```

apiVersion: v1
kind: Pod
metadata:
  name: s1
spec:
  affinity: ①
    nodeAffinity:
      preferredDuringSchedulingIgnoredDuringExecution: ②
        - weight: ③
          preference:
            matchExpressions:
              - key: e2e-az-name ④
                values:
                  - e2e-az3
            operator: In ⑤
#...
```

①

Pod のアフィニティーを追加します。

②

`preferredDuringSchedulingIgnoredDuringExecution` パラメーターを設定します。

③

ノードの重みを数字の 1-100 で指定します。最も高い重みを持つノードが優先されます。

④

満たす必要のある `key` および `values` を指定します。新規 Pod を編集したノードにスケジュールする必要がある場合、ノードのラベルと同じ `key` および `values` パラメーターを使用します。

⑤

Operator を指定します。演算子は `In`、`NotIn`、`Exists`、または `DoesNotExist` にすることができます。たとえば、演算子 `In` を使用してラベルをノードで必要になるようにします。

- b. Pod を作成します。

```
$ oc create -f <file-name>.yaml
```

4.3.4. ノードのアフィニティールールの例

以下の例は、ノードのアフィニティーを示しています。

4.3.4.1. 一致するラベルを持つノードのアフィニティー

以下の例は、一致するラベルを持つノードと Pod のノードのアフィニティーを示しています。

- Node1 ノードにはラベル `zone:us` があります。

```
$ oc label node node1 zone=us
```

ヒント

あるいは、以下の YAML を適用してラベルを追加できます。

```
kind: Node
apiVersion: v1
metadata:
  name: <node_name>
  labels:
    zone: us
#...
```

- `pod-s1 pod` にはノードアフィニティーの `required` (必須) ルールの下に `zone` と `us` のキー/値のペアがあります。

```
$ cat pod-s1.yaml
```

出力例

```
apiVersion: v1
kind: Pod
```

```

metadata:
  name: pod-s1
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
    - image: "docker.io/ocpqe/hello-pod"
      name: hello-pod
      securityContext:
        allowPrivilegeEscalation: false
      capabilities:
        drop: [ALL]
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
          - matchExpressions:
              - key: "zone"
                operator: In
                values:
                  - us
#...

```

- pod-s1 pod は Node1 でスケジュールできます。

```
$ oc get pod -o wide
```

出力例

```

NAME      READY   STATUS    RESTARTS  AGE   IP      NODE
pod-s1    1/1     Running   0          4m   IP1     node1

```

4.3.4.2. 一致するラベルのないノードのアフィニティー

以下の例は、一致するラベルを持たないノードと Pod のノードのアフィニティーを示しています。

- Node1 ノードにはラベル `zone:emea` があります。

```
$ oc label node node1 zone=emea
```

ヒント

あるいは、以下の YAML を適用してラベルを追加できます。

```
kind: Node
apiVersion: v1
metadata:
  name: <node_name>
  labels:
    zone: emea
#...
```

- `pod-s1 pod` にはノードアフィニティーの `required` (必須) ルールの下に `zone` と `us` のキー/値のペアがあります。

```
$ cat pod-s1.yaml
```

出力例

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-s1
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
    - image: "docker.io/ocpqe/hello-pod"
      name: hello-pod
      securityContext:
        allowPrivilegeEscalation: false
      capabilities:
        drop: [ALL]
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
          - matchExpressions:
              - key: "zone"
```

```

operator: In
values:
- us
#...

```

- pod-s1 pod は Node1 でスケジュールすることができません。

```
$ oc describe pod pod-s1
```

出力例

```

...
Events:
  FirstSeen LastSeen Count From              SubObjectPath Type           Reason
  ----
  1m         33s         8   default-scheduler Warning        FailedScheduling No nodes are
available that match all of the following predicates:: MatchNodeSelector (1).

```

4.4. POD のオーバーコミットノードへの配置

オーバーコミットとは、コンテナの計算リソース要求と制限の合計が、そのシステムで利用できるリソースを超えた状態のことです。オーバーコミットは、容量に対して保証されたパフォーマンスのトレードオフが許容可能である開発環境において、望ましいことがあります。

要求および制限により、管理者はノードでのリソースのオーバーコミットを許可し、管理できます。スケジューラーは、要求を使用してコンテナをスケジュールし、最小限のサービス保証を提供します。制限は、ノード上で消費されるコンピュートリソースの量を制限します。

4.4.1. オーバーコミットについて

要求および制限により、管理者はノードでのリソースのオーバーコミットを許可し、管理できます。スケジューラーは、要求を使用してコンテナをスケジュールし、最小限のサービス保証を提供します。制限は、ノード上で消費されるコンピュートリソースの量を制限します。

Red Hat OpenShift Service on AWS の管理者は、開発者用のコンテナに設定された要求と制限の比率をオーバーライドするようにマスターを設定することで、オーバーコミットのレベルを制御し、ノード上のコンテナ密度を管理できます。この設定を、制限とデフォルトを指定するプロジェクトごとの `LimitRange` と共に使用することで、オーバーコミットを必要なレベルに設定できるようコンテナの制限と要求を調整することができます。



注記

コンテナに制限が設定されていない場合には、これらの上書きは影響を与えません。デフォルトの制限で (個別プロジェクトごとに、またはプロジェクトテンプレートを介して) `LimitRange` オブジェクトを作成し、上書きが適用されるようにします。

上書き後も、コンテナの制限および要求は、プロジェクトのいずれかの `LimitRange` オブジェクトで引き続き検証される必要があります。たとえば、開発者が最小限度に近い制限を指定し、要求を最小限度よりも低い値に上書きすることで、Pod が禁止される可能性があります。この最適でないユーザーエクスペリエンスについては、今後の作業で対応する必要がありますが、現時点ではこの機能および `LimitRange` オブジェクトを注意して設定してください。

4.4.2. ノードのオーバーコミットについて

オーバーコミット環境では、最適なシステム動作を提供できるようにノードを適切に設定する必要があります。

ノードが起動すると、メモリー管理用のカーネルの調整可能なフラグが適切に設定されます。カーネルは、物理メモリーが不足しない限り、メモリーの割り当てに失敗することはありません。

この動作を確実にするために、Red Hat OpenShift Service on AWS は、`vm.overcommit_memory` パラメーターを 1 に設定し、デフォルトのオペレーティングシステム設定をオーバーライドすることで、常にメモリーをオーバーコミットするようにカーネルを設定します。

また、Red Hat OpenShift Service on AWS は、`vm.panic_on_oom` パラメーターを 0 に設定することで、メモリー不足時のカーネルパニックを回避します。0 の設定は、Out of Memory (OOM) 状態のときに `oom_killer` を呼び出すようカーネルに指示します。これにより、優先順位に基づいてプロセスを強制終了します。

現在の設定は、ノードに以下のコマンドを実行して表示できます。

```
$ sysctl -a |grep commit
```

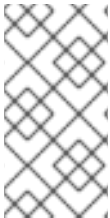
出力例

```
#...  
vm.overcommit_memory = 0  
#...
```

```
$ sysctl -a |grep panic
```

出力例

```
#...  
vm.panic_on_oom = 0  
#...
```



注記

上記のフラグはノード上にすでに設定されているはずであるため、追加のアクションは不要です。

各ノードに対して以下の設定を実行することもできます。

- CPU CFS クォータを使用した CPU 制限の無効化または実行
- システムプロセスのリソース予約
- Quality of Service (QoS) 層でのメモリー予約

4.5. ノードテイントを使用した POD 配置の制御

テイントおよび容認 (Toleration) により、ノードはノード上でスケジュールする必要のある (またはスケジュールすべきでない) Pod を制御できます。

4.5.1. テイントおよび容認 (Toleration) について

テイントにより、ノードは Pod に一致する容認がない場合に Pod のスケジュールを拒否することができます。

テイントは Node 仕様 (NodeSpec) でノードに適用され、容認は Pod 仕様 (PodSpec) で Pod に適用されます。テイントをノードに適用する場合、スケジューラーは Pod がテイントを容認しない限り、Pod をそのノードに配置することができません。

ノード仕様のテイントの例

```
apiVersion: v1
kind: Node
metadata:
  name: my-node
#...
spec:
  taints:
  - effect: NoExecute
    key: key1
    value: value1
#...
```

Pod 仕様での容認の例

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
#...
spec:
  tolerations:
  - key: "key1"
    operator: "Equal"
    value: "value1"
    effect: "NoExecute"
    tolerationSeconds: 3600
#...
```

-

テイントおよび容認は、**key**、**value**、および **effect** で構成されます。

表4.1 テイントおよび容認コンポーネント

| パラメーター | 説明 | | | | | | |
|----------------------------------|---|----------------------------------|--|-------------------------|--|------------------|--|
| key | key には、253 文字までの文字列を使用できます。キーは文字または数字で開始する必要があり、文字、数字、ハイフン、ドットおよびアンダースコアを含めることができます。 | | | | | | |
| value | value には、63 文字までの文字列を使用できます。値は文字または数字で開始する必要があり、文字、数字、ハイフン、ドットおよびアンダースコアを含めることができます。 | | | | | | |
| effect | effect は以下のいずれかにすることができます。 <table border="1" data-bbox="518 958 1428 1713"> <tbody> <tr> <td>NoSchedule ^[1]</td> <td> <ul style="list-style-type: none"> • テイントに一致しない新規 Pod はノードにスケジュールされません。 • ノードの既存 Pod はそのままになります。 </td> </tr> <tr> <td>PreferNoSchedule</td> <td> <ul style="list-style-type: none"> • テイントに一致しない新規 Pod はノードにスケジュールされる可能性があります。スケジューラーはスケジュールしないようにします。 • ノードの既存 Pod はそのままになります。 </td> </tr> <tr> <td>NoExecute</td> <td> <ul style="list-style-type: none"> • テイントに一致しない新規 Pod はノードにスケジュールできません。 • 一致する容認を持たないノードの既存 Pod は削除されます。 </td> </tr> </tbody> </table> | NoSchedule ^[1] | <ul style="list-style-type: none"> • テイントに一致しない新規 Pod はノードにスケジュールされません。 • ノードの既存 Pod はそのままになります。 | PreferNoSchedule | <ul style="list-style-type: none"> • テイントに一致しない新規 Pod はノードにスケジュールされる可能性があります。スケジューラーはスケジュールしないようにします。 • ノードの既存 Pod はそのままになります。 | NoExecute | <ul style="list-style-type: none"> • テイントに一致しない新規 Pod はノードにスケジュールできません。 • 一致する容認を持たないノードの既存 Pod は削除されます。 |
| NoSchedule ^[1] | <ul style="list-style-type: none"> • テイントに一致しない新規 Pod はノードにスケジュールされません。 • ノードの既存 Pod はそのままになります。 | | | | | | |
| PreferNoSchedule | <ul style="list-style-type: none"> • テイントに一致しない新規 Pod はノードにスケジュールされる可能性があります。スケジューラーはスケジュールしないようにします。 • ノードの既存 Pod はそのままになります。 | | | | | | |
| NoExecute | <ul style="list-style-type: none"> • テイントに一致しない新規 Pod はノードにスケジュールできません。 • 一致する容認を持たないノードの既存 Pod は削除されます。 | | | | | | |

| パラメーター | 説明 | |
|----------|--------|--|
| operator | Equal | key/value/effect パラメーターは一致する必要があります。これはデフォルトになります。 |
| | Exists | key/effect パラメーターは一致する必要があります。いずれかに一致する value パラメーターを空のままにする必要があります。 |

1. **NoSchedule** テイントをコントロールプレーンノードに追加する場合、ノードには、デフォルトで追加される `node-role.kubernetes.io/master=:NoSchedule` テイントが必要です。

以下に例を示します。

```

apiVersion: v1
kind: Node
metadata:
  annotations:
    machine.openshift.io/machine: openshift-machine-api/ci-ln-62s7gtb-f76d1-v8jxv-master-0
    machineconfiguration.openshift.io/currentConfig: rendered-master-cdc1ab7da414629332cc4c3926e6e59c
    name: my-node
#...
spec:
  taints:
    - effect: NoSchedule
      key: node-role.kubernetes.io/master
#...

```

容認はテイントと一致します。

- operator パラメーターが **Equal** に設定されている場合:
 - **key** パラメーターは同じになります。

- value パラメーターは同じになります。
- effect パラメーターは同じになります。
- operator パラメーターが **Exists** に設定されている場合:
 - key パラメーターは同じになります。
 - effect パラメーターは同じになります。

次のテイントは、Red Hat OpenShift Service on AWS に組み込まれています。

- **node.kubernetes.io/not-ready**: ノードは準備状態ではありません。これはノード条件 **Ready=False** に対応します。
- **node.kubernetes.io/unreachable**: ノードはノードコントローラーから到達不能です。これはノード条件 **Ready=Unknown** に対応します。
- **node.kubernetes.io/memory-pressure**: ノードにはメモリー不足の問題が発生しています。これはノード条件 **MemoryPressure=True** に対応します。
- **node.kubernetes.io/disk-pressure**: ノードにはディスク不足の問題が発生しています。これはノード条件 **DiskPressure=True** に対応します。
- **node.kubernetes.io/network-unavailable**: ノードのネットワークは使用できません。
- **node.kubernetes.io/unschedulable**: ノードはスケジュールが行えません。
- **node.cloudprovider.kubernetes.io/uninitialized**: ノードコントローラーが外部のクラウドプロバイダーを使用して起動すると、このテイントはノード上に設定され、使用不可能と

マークされます。cloud-controller-manager のコントローラーがこのノードを初期化した後に、kubelet がこのテイントを削除します。

- **node.kubernetes.io/pid-pressure:** ノードが pid 不足の状態です。これはノード条件 PIDPressure=True に対応します。



重要

Red Hat OpenShift Service on AWS では、デフォルトの pid.available evictionHard は設定されません。

4.5.1.1. Pod のエビクションを遅延させる容認期間 (秒数) の使用方法

Pod 仕様または MachineSet に tolerationSeconds パラメーターを指定して、Pod がエビクションされる前にノードにバインドされる期間を指定できます。effect が NoExecute のテイントがノードに追加される場合、テイントを容認する Pod に tolerationSeconds パラメーターがある場合、Pod は期限切れになるまでエビクトされません。

出力例

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
#...
spec:
  tolerations:
  - key: "key1"
    operator: "Equal"
    value: "value1"
    effect: "NoExecute"
    tolerationSeconds: 3600
#...
```

ここで、この Pod が実行中であるものの、一致する容認がない場合、Pod は 3,600 秒間バインドされたままとなり、その後にエビクトされます。テイントが期限前に削除される場合、Pod はエビクトされません。

4.5.1.2. 複数のテイントの使用法

複数のテイントを同じノードに、複数の容認を同じ Pod に配置することができます。Red Hat OpenShift Service on AWS は複数のテイントと容認を以下のように処理します。

1. Pod に一致する容認のあるテイントを処理します。
2. 残りの一致しないテイントは Pod について以下の effect を持ちます。
 - NoSchedule effect を持つ一致しないテイントが 1 つ以上ある場合、Red Hat OpenShift Service on AWS はそのノードに Pod をスケジュールできません。
 - NoSchedule effect を持つ一致しないテイントはないが、PreferNoSchedule effect を持つ一致しないテイントが 1 つ以上ある場合、Red Hat OpenShift Service on AWS はノードに Pod をスケジュールしないように試みます。
 - NoExecute effect を持つ一致しないテイントが 1 つ以上ある場合、ノード上で Pod がすでに実行されている場合は Red Hat OpenShift Service on AWS によって Pod がノードから削除され、ノード上で Pod がまだ実行されていない場合は Pod がノードにスケジュールされません。
 - テイントを容認しない Pod はすぐにエビクトされます。
 - Pod の仕様に tolerationSeconds を指定せずにテイントを容認する Pod は永久にバインドされたままになります。
 - 指定された tolerationSeconds を持つテイントを容認する Pod は指定された期間バインドされます。

以下に例を示します。

- 以下のテイントをノードに追加します。

```
$ oc adm taint nodes node1 key1=value1:NoSchedule
```

```
$ oc adm taint nodes node1 key1=value1:NoExecute
```

```
$ oc adm taint nodes node1 key2=value2:NoSchedule
```

- Pod には以下の容認があります。

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
#...
spec:
  tolerations:
  - key: "key1"
    operator: "Equal"
    value: "value1"
    effect: "NoSchedule"
  - key: "key1"
    operator: "Equal"
    value: "value1"
    effect: "NoExecute"
#...
```

この場合、3つ目のテイントに一致する容認がないため、Pod はノードにスケジュールできません。Pod はこのテイントの追加時にノードですでに実行されている場合は実行が継続されます。3つ目のテイントは3つのテイントの中でPod で容認されない唯一のテイントであるためです。

4.5.1.3. Pod のスケジューリングとノードの状態 (Taint Nodes By Condition) について

Taint Nodes By Condition (状態別のノードへのテイント) 機能はデフォルトで有効にされており、これはメモリ不足やディスク不足などの状態を報告するノードを自動的にテイントします。ノードが状態を報告すると、その状態が解消するまでテイントが追加されます。テイントに NoSchedule の effect がある場合、ノードが一致する容認を持つまでそのノードに Pod をスケジュールすることはできません。

スケジューラーは、Pod をスケジュールする前に、ノードでこれらのテイントの有無をチェックします。テイントがある場合、Pod は別のノードにスケジュールされます。スケジューラーは実際のノードの状態ではなくテイントをチェックするので、適切な Pod 容認を追加して、スケジューラーがこのようなノードの状態を無視するように設定します。

デーモンセットコントローラーは、以下の容認をすべてのデーモンに自動的に追加し、下位互換性を確保します。

- node.kubernetes.io/memory-pressure

- `node.kubernetes.io/disk-pressure`
- `node.kubernetes.io/unschedulable` (1.10 以降)
- `node.kubernetes.io/network-unavailable` (ホストネットワークのみ)

デーモンセットには任意の容認を追加することも可能です。



注記

コントロールプレーンは、QoS クラスを持つ Pod に `node.kubernetes.io/memory-pressure` 容認も追加します。これは、Kubernetes が **Guaranteed** または **Burstable** QoS クラスで Pod を管理するためです。新しい **BestEffort** Pod は、影響を受けるノードにスケジュールされません。

4.5.1.4. Pod の状態別エビクションについて (Taint-Based Eviction)

Taint-Based Eviction 機能はデフォルトで有効にされており、これは `not-ready` および `unreachable` などの特定の状態にあるノードから Pod をエビクトします。ノードがこうした状態のいずれかになると、Red Hat OpenShift Service on AWS はテイントをノードに自動的に追加して、Pod のエビクトおよび別のノードでの再スケジュールを開始します。

Taint Based Eviction には `NoExecute` の effect があり、そのテイントを容認しない Pod はすぐにエビクトされ、これを容認する Pod はエビクトされません (Pod が `tolerationSeconds` パラメーターを使用しない場合に限りです)。

`tolerationSeconds` パラメーターを使用すると、ノード状態が設定されたノードに Pod がどの程度の期間バインドされるかを指定することができます。`tolerationSeconds` の期間後もこの状態が続くと、テイントはノードに残り続け、一致する容認を持つ Pod はエビクトされます。`tolerationSeconds` の期間前にこの状態が解消される場合、一致する容認を持つ Pod は削除されません。

値なしで `tolerationSeconds` パラメーターを使用する場合、Pod は `not ready`(準備未完了) および `unreachable`(到達不能) のノードの状態が原因となりエビクトされることはありません。

注記

Red Hat OpenShift Service on AWS は、レートが制限された方法で Pod をエビク
トし、マスターがノードからパーティション化される場合などのシナリオで発生する大
規模な Pod エビクションを防ぎます。

デフォルトでは、特定のゾーン内のノードの 55% 以上が異常である場合、ノードラ
イフサイクルコントローラーはそのゾーンの状態を **PartialDisruption** に変更し、Pod
の削除率が低下します。この状態の小さなクラスター (デフォルトでは 50 ノード以下)
の場合、このゾーンのノードはテイントされず、排除が停止されます。

詳細については、Kubernetes ドキュメントの [Rate limits on eviction](#) を参照して
ください。

Red Hat OpenShift Service on AWS は、`tolerationSeconds=300` を使用して
`node.kubernetes.io/not-ready` および `node.kubernetes.io/unreachable` の容認を自動的に追加しま
す。ただし、Pod 設定でどちらかの容認が指定されている場合を除きます。

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
#...
spec:
  tolerations:
  - key: node.kubernetes.io/not-ready
    operator: Exists
    effect: NoExecute
    tolerationSeconds: 300 1
  - key: node.kubernetes.io/unreachable
    operator: Exists
    effect: NoExecute
    tolerationSeconds: 300
#...
```

1

これらの容認は、ノード状態の問題のいずれかが検出された後、デフォルトの Pod 動作のパ
インドを 5 分間維持できるようにします。

これらの容認は必要に応じて設定できます。たとえば、アプリケーションに多数のローカル状態が
ある場合、ネットワークのパーティション化などに伴い、Pod をより長い時間ノードにバインドさせる
必要があるかもしれません。これにより、パーティションを回復させることができ、Pod のエビク
ションを回避できます。

デーモンセットによって起動する Pod は、`tolerationSeconds` が指定されない以下のテイントの `NoExecute` 容認を使用して作成されます。

- `node.kubernetes.io/unreachable`
- `node.kubernetes.io/not-ready`

その結果、デーモンセット Pod は、これらのノードの状態が原因でエビクトされることはありません。

4.5.1.5. すべてのテイントの許容

ノードは、`operator: "Exists"` 容認を `key` および `value` パラメーターなしで追加することですべてのテイントを容認するように Pod を設定できます。この容認のある Pod はテイントを持つノードから削除されません。

すべてのテイントを容認するための Pod 仕様

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
#...
spec:
  tolerations:
    - operator: "Exists"
#...
```

4.5.2. テイントおよび容認 (Toleration) の追加

容認を Pod に、テイントをノードに追加することで、ノードはノード上でスケジュールする必要のある (またはスケジュールすべきでない) Pod を制御できます。既存の Pod およびノードの場合、最初に容認を Pod に追加してからテイントをノードに追加して、容認を追加する前に Pod がノードから削除されないようにする必要があります。

手順

1. Pod 仕様を tolerations スタンザを含めるように編集して、容認を Pod に追加します。

Equal 演算子を含む Pod 設定ファイルのサンプル

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
#...
spec:
  tolerations:
  - key: "key1" ①
    value: "value1"
    operator: "Equal"
    effect: "NoExecute"
    tolerationSeconds: 3600 ②
#...
```

①

テイントおよび容認コンポーネントの表で説明されている toleration パラメーターです。

②

tolerationSeconds パラメーターは、エビクトする前に Pod をどの程度の期間ノードにバインドさせるかを指定します。

以下に例を示します。

Exists 演算子を含む Pod 設定ファイルのサンプル

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
#...
spec:
  tolerations:
```

```
- key: "key1"  
  operator: "Exists" ①  
  effect: "NoExecute"  
  tolerationSeconds: 3600  
#...
```

①

Exists Operator は value を取りません。

この例では、テイントを、キー key1、値 value1、およびテイント effect NoExecute を持つ node1 にテイントを配置します。

2.

テイントおよび容認コンポーネントの表で説明されているパラメーターと共に以下のコマンドを使用してテイントをノードに追加します。

```
$ oc adm taint nodes <node_name> <key>=<value>:<effect>
```

以下に例を示します。

```
$ oc adm taint nodes node1 key1=value1:NoExecute
```

このコマンドは、キー key1、値 value1、および effect NoExecute を持つテイントを node1 に配置します。

注記

NoSchedule テイントをコントロールプレーンノードに追加する場合、ノードには、デフォルトで追加される `node-role.kubernetes.io/master=:NoSchedule` テイントが必要です。

以下に例を示します。

```
apiVersion: v1
kind: Node
metadata:
  annotations:
    machine.openshift.io/machine: openshift-machine-api/ci-ln-62s7gtb-
f76d1-v8jxv-master-0
    machineconfiguration.openshift.io/currentConfig: rendered-master-
cdc1ab7da414629332cc4c3926e6e59c
    name: my-node
#...
spec:
  taints:
    - effect: NoSchedule
      key: node-role.kubernetes.io/master
#...
```

Pod の容認はノードのテイントに一致します。いずれかの容認のある Pod は node1 にスケジューリングできます。

4.5.2.1. コンピュートマシンセットを使用したテイントおよび容認の追加

コンピュートマシンセットを使用してテイントをノードに追加できます。MachineSet オブジェクトに関連付けられるすべてのノードがテイントで更新されます。容認は、ノードに直接追加されたテイントと同様に、コンピュートマシンセットによって追加されるテイントに応答します。

手順

1. Pod 仕様を `tolerations` スタンザを含めるように編集して、容認を Pod に追加します。

Equal 演算子を含む Pod 設定ファイルのサンプル

```
apiVersion: v1
kind: Pod
metadata:
```

```
name: my-pod
#...
spec:
  tolerations:
  - key: "key1" ①
    value: "value1"
    operator: "Equal"
    effect: "NoExecute"
    tolerationSeconds: 3600 ②
#...
```

①

テイントおよび容認コンポーネントの表で説明されている toleration パラメーターです。

②

tolerationSeconds パラメーターは、エビクトする前に Pod をどの程度の期間ノードにバインドさせるかを指定します。

以下に例を示します。

Exists 演算子を含む Pod 設定ファイルのサンプル

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
#...
spec:
  tolerations:
  - key: "key1"
    operator: "Exists"
    effect: "NoExecute"
    tolerationSeconds: 3600
#...
```

2.

テイントを `MachineSet` オブジェクトに追加します。

a.

テイントを付けるノードの `MachineSet` YAML を編集するか、新規 `MachineSet` オブジェクトを作成できます。

```
$ oc edit machineset <machineset>
```

b.

テイントを `spec.template.spec` セクションに追加します。

コンピューターマシンセット仕様のテイントの例

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
  name: my-machineset
#...
spec:
#...
  template:
#...
    spec:
      taints:
      - effect: NoExecute
        key: key1
        value: value1
#...
```

この例では、キー `key1`、値 `value1`、およびテイント `effect NoExecute` を持つテイントをノードに配置します。

c.

コンピューターマシンセットを 0 にスケールダウンします。

```
$ oc scale --replicas=0 machineset <machineset> -n openshift-machine-api
```

ヒント

または、以下の YAML を適用してコンピュータマシンセットをスケーリングすることもできます。

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
  name: <machineset>
  namespace: openshift-machine-api
spec:
  replicas: 0
```

マシンが削除されるまで待機します。

d.

コンピュータマシンセットを随時スケールアップします。

```
$ oc scale --replicas=2 machineset <machineset> -n openshift-machine-api
```

または、以下を実行します。

```
$ oc edit machineset <machineset> -n openshift-machine-api
```

マシンが起動するまで待ちます。テイントは MachineSet オブジェクトに関連付けられたノードに追加されます。

4.5.2.2. テイントおよび容認 (Toleration) 使用してユーザーをノードにバインドする

ノードのセットを特定のユーザーセットによる排他的な使用のために割り当てる必要がある場合、容認をそれらの Pod に追加します。次に、対応するテイントをそれらのノードに追加します。容認が設定された Pod は、テイントが付けられたノードまたはクラスター内の他のノードを使用できます。

Pod がテイントが付けられたノードのみにスケジュールされるようにするには、ラベルを同じノードセットに追加し、ノードのアフィニティーを Pod に追加し、Pod がそのラベルの付いたノードのみにスケジュールできるようにします。

手順

ノードをユーザーの使用可能な唯一のノードとして設定するには、以下を実行します。

1. 対応するテイントをそれらのノードに追加します。

以下に例を示します。

```
$ oc adm taint nodes node1 dedicated=groupName:NoSchedule
```

ヒント

または、以下の YAML を適用してテイントを追加できます。

```
kind: Node
apiVersion: v1
metadata:
  name: my-node
#...
spec:
  taints:
    - key: dedicated
      value: groupName
      effect: NoSchedule
#...
```

2. カスタム受付コントローラーを作成して容認を Pod に追加します。

4.5.2.3. ノードセレクターおよび容認を使用したプロジェクトの作成

ノードセレクターおよび容認 (アノテーションとして設定されたもの) を使用するプロジェクトを作成して、Pod の特定のノードへの配置を制御できます。プロジェクトで作成された後続のリソースは、容認に一致するテイントを持つノードでスケジュールされます。

前提条件

- コンピュートマシンセットを使用するか、ノードを直接編集して、ノード選択のラベルが 1 つ以上のノードに追加されている。
- コンピュートマシンセットを使用するか、ノードを直接編集することによって、テイントが 1 つ以上のノードに追加されました。

手順

1. `metadata.annotations` セクションにノードセクターおよび容認を指定して、Project リソース定義を作成します。

project.yaml ファイルの例

```
kind: Project
apiVersion: project.openshift.io/v1
metadata:
  name: <project_name> ❶
  annotations:
    openshift.io/node-selector: '<label>' ❷
    scheduler.alpha.kubernetes.io/defaultTolerations: >-
      [{"operator": "Exists", "effect": "NoSchedule", "key":
        "<key_name>"}] ❸
  ]
```

❶

プロジェクト名。

❷

デフォルトのノードセクターラベル。

❸

テイントおよび容認コンポーネント の表で説明されている `toleration` パラメーターです。この例では、`NoSchedule` の `effect` を使用します。これにより、ノード上の既存の Pod はそのまま残り、`Exists Operator` は値を取得しません。

2. `oc apply` コマンドを使用してプロジェクトを作成します。

```
$ oc apply -f project.yaml
```

`<project_name> namespace` で作成された後続のリソースは指定されたノードにスケジュールされます。

関連情報

- **ノードに手動で、または コンピュートマシンセットを使用**してテイントと容認を追加する

4.5.2.4. テイントおよび容認 (Toleration) を使用して特殊ハードウェアを持つノードを制御する

ノードの小規模なサブセットが特殊ハードウェアを持つクラスターでは、テイントおよび容認 (Toleration) を使用して、特殊ハードウェアを必要としない Pod をそれらのノードから切り離し、特殊ハードウェアを必要とする Pod をそのままにすることができます。また、特殊ハードウェアを必要とする Pod に対して特定のノードを使用することを要求することもできます。

これは、特殊ハードウェアを必要とする Pod に容認を追加し、特殊ハードウェアを持つノードにテイントを付けることで実行できます。

手順

特殊ハードウェアを持つノードが特定の Pod 用に予約されるようにするには、以下を実行します。

1. 容認を特別なハードウェアを必要とする Pod に追加します。

以下に例を示します。

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
#...
spec:
  tolerations:
    - key: "disktype"
      value: "ssd"
      operator: "Equal"
      effect: "NoSchedule"
      tolerationSeconds: 3600
#...
```

2. 以下のコマンドのいずれかを使用して、特殊ハードウェアを持つノードにテイントを設定します。

```
$ oc adm taint nodes <node-name> disktype=ssd:NoSchedule
```

または、以下を実行します。

```
$ oc adm taint nodes <node-name> disktype=ssd:PreferNoSchedule
```

ヒント

または、以下の YAML を適用してテイントを追加できます。

```
kind: Node
apiVersion: v1
metadata:
  name: my_node
#...
spec:
  taints:
  - key: disktype
    value: ssd
    effect: PreferNoSchedule
#...
```

4.5.3. テイントおよび容認 (Toleration) の削除

必要に応じてノードからテイントを、Pod から容認をそれぞれ削除できます。最初に容認を Pod に追加してからテイントをノードに追加して、容認を追加する前に Pod がノードから削除されないようにする必要があります。

手順

テイントおよび容認 (Toleration) を削除するには、以下を実行します。

1. ノードからテイントを削除するには、以下を実行します。

```
$ oc adm taint nodes <node-name> <key>-
```

以下に例を示します。

```
$ oc adm taint nodes ip-10-0-132-248.ec2.internal key1-
```

出力例


```
node/ip-10-0-132-248.ec2.internal untainted
```

2.

Pod から容認を削除するには、容認を削除するための Pod 仕様を編集します。

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
#...
spec:
  tolerations:
  - key: "key2"
    operator: "Exists"
    effect: "NoExecute"
    tolerationSeconds: 3600
#...
```

4.6. ノードセクターの使用による特定ノードへの POD の配置

ノードセクターは、ノードのカスタムラベルと Pod で指定されるセクターを使用して定義されるキー/値のペアのマップを指定します。

Pod がノードで実行する要件を満たすには、Pod にはノードのラベルと同じキー/値のペアがなければなりません。

4.6.1. ノードセクターについて

Pod でノードセクターを使用し、ノードでラベルを使用して、Pod がスケジュールされる場所を制御できます。ノードセクターを使用すると、Red Hat OpenShift Service on AWS は一致するラベルが含まれるノード上に Pod をスケジュールします。

ノードセクターを使用して特定の Pod を特定のノードに配置し、クラスタースコープのノードセクターを使用して特定ノードの新規 Pod をクラスター内の任意の場所に配置し、プロジェクトノードを使用して新規 Pod を特定ノードのプロジェクトに配置できます。

たとえば、クラスター管理者は、作成するすべての Pod にノードセクターを追加して、アプリケーション開発者が地理的に最も近い場所にあるノードにのみ Pod をデプロイできるインフラストラクチャーを作成できます。この例では、クラスターは 2 つのリージョンに分散する 5 つのデータセンターで構成されます。米国では、ノードに us-east、us-central、または us-west のラベルを付けま

す。アジア太平洋リージョン (APAC) では、ノードに `apac-east` または `apac-west` のラベルを付けます。開発者は、Pod がこれらのノードにスケジュールされるように、作成する Pod にノードセクターを追加できます。

Pod オブジェクトにノードセクターが含まれる場合でも、一致するラベルを持つノードがない場合、Pod はスケジュールされません。



重要

同じ Pod 設定でノードセクターとノードのアフィニティーを使用している場合は、以下のルールが Pod のノードへの配置を制御します。

- `nodeSelector` と `nodeAffinity` の両方を設定する場合、Pod が候補ノードでスケジュールされるにはどちらの条件も満たしている必要があります。
- `nodeAffinity` タイプに関連付けられた複数の `nodeSelectorTerms` を指定する場合、`nodeSelectorTerms` のいずれかが満たされている場合に Pod をノードにスケジュールすることができます。
- `nodeSelectorTerms` に関連付けられた複数の `matchExpressions` を指定する場合、すべての `matchExpressions` が満たされている場合にのみ Pod をノードにスケジュールすることができます。

特定の Pod およびノードのノードセクター

ノードセクターおよびラベルを使用して、特定の Pod がスケジュールされるノードを制御できます。

ノードセクターおよびラベルを使用するには、まずノードにラベルを付けて Pod がスケジュール解除されないようにしてから、ノードセクターを Pod に追加します。



注記

ノードセクターを既存のスケジュールされている Pod に直接追加することはできません。デプロイメント設定などの Pod を制御するオブジェクトにラベルを付ける必要があります。

たとえば、以下の Node オブジェクトには `region: east` ラベルがあります。

ラベルを含む Node オブジェクトのサンプル

```
kind: Node
apiVersion: v1
metadata:
  name: ip-10-0-131-14.ec2.internal
  selfLink: /api/v1/nodes/ip-10-0-131-14.ec2.internal
  uid: 7bc2580a-8b8e-11e9-8e01-021ab4174c74
  resourceVersion: '478704'
  creationTimestamp: '2019-06-10T14:46:08Z'
  labels:
    kubernetes.io/os: linux
    topology.kubernetes.io/zone: us-east-1a
    node.openshift.io/os_version: '4.5'
    node-role.kubernetes.io/worker: ""
    topology.kubernetes.io/region: us-east-1
    node.openshift.io/os_id: rhcos
    node.kubernetes.io/instance-type: m4.large
    kubernetes.io/hostname: ip-10-0-131-14
    kubernetes.io/arch: amd64
    region: east ❶
    type: user-node
#...
```

❶

Pod ノードセレクトターに一致するラベル。

Pod には `type: user-node,region: east` ノードセレクトターがあります。

ノードセレクトターが含まれる Pod オブジェクトのサンプル

```
apiVersion: v1
kind: Pod
metadata:
  name: s1
#...
spec:
```

```
nodeSelector: 1
  region: east
  type: user-node
#...
```

1

ノードトラベルに一致するノードセレクター。ノードには、各ノードセレクターのラベルが必要です。

サンプル Pod 仕様を使用して Pod を作成する場合、これはサンプルノードでスケジュールできます。

クラスタースコープのデフォルトノードセレクター

デフォルトのクラスタースコープのノードセレクターを使用する場合、クラスターで Pod を作成すると、Red Hat OpenShift Service on AWS はデフォルトのノードセレクターを Pod に追加し、一致するラベルのあるノードで Pod をスケジュールします。

たとえば、以下の Scheduler オブジェクトにはデフォルトのクラスタースコープの `region=east` および `type=user-node` ノードセレクターがあります。

スケジューラー Operator カスタムリソースの例

```
apiVersion: config.openshift.io/v1
kind: Scheduler
metadata:
  name: cluster
#...
spec:
  defaultNodeSelector: type=user-node,region=east
#...
```

クラスター内のノードには `type=user-node,region=east` ラベルがあります。

Node オブジェクトの例

```

apiVersion: v1
kind: Node
metadata:
  name: ci-ln-qg1il3k-f76d1-hlmhl-worker-b-df2s4
  #...
  labels:
    region: east
    type: user-node
  #...

```

ノードセレクターを持つ Pod オブジェクトの例

```

apiVersion: v1
kind: Pod
metadata:
  name: s1
  #...
spec:
  nodeSelector:
    region: east
  #...

```

サンプルクラスターでサンプル Pod 仕様を使用して Pod を作成する場合、Pod はクラスター スコープのノードセレクターで作成され、ラベルが付けられたノードにスケジュールされます。

ラベルが付けられたノード上の Pod を含む Pod リストの例

```

NAME      READY STATUS RESTARTS AGE IP      NODE
NOMINATED NODE READINESS GATES
pod-s1 1/1   Running 0      20s 10.131.2.6 ci-ln-qg1il3k-f76d1-hlmhl-worker-b-df2s4
<none>      <none>

```



注記

Pod を作成するプロジェクトにプロジェクトノードセレクターがある場合、そのセレクターはクラスタースコープのセレクターよりも優先されます。Pod にプロジェクトノードセレクターがない場合、Pod は作成されたり、スケジュールされたりしません。

プロジェクトノードセレクター

プロジェクトノードセレクターを使用する場合、このプロジェクトで Pod を作成すると、Red Hat OpenShift Service on AWS はノードセレクターを Pod に追加し、一致するラベルを持つノードで Pod をスケジュールします。クラスタースコープのデフォルトノードセレクターがない場合、プロジェクトノードセレクターが優先されます。

たとえば、以下のプロジェクトには `region=east` ノードセレクターがあります。

Namespace オブジェクトの例

```
apiVersion: v1
kind: Namespace
metadata:
  name: east-region
  annotations:
    openshift.io/node-selector: "region=east"
#...
```

以下のノードには `type=user-node,region=east` ラベルがあります。

Node オブジェクトの例

```
apiVersion: v1
kind: Node
metadata:
  name: ci-ln-qg1il3k-f76d1-hlmhl-worker-b-df2s4
#...
labels:
```

```

region: east
type: user-node
#...

```

Pod をこのサンプルプロジェクトでサンプル Pod 仕様を使用して作成する場合、Pod はプロジェクトノードセクターで作成され、ラベルが付けられたノードにスケジュールされます。

Pod オブジェクトの例

```

apiVersion: v1
kind: Pod
metadata:
  namespace: east-region
#...
spec:
  nodeSelector:
    region: east
    type: user-node
#...

```

ラベルが付けられたノード上の Pod を含む Pod リストの例

```

NAME      READY  STATUS   RESTARTS  AGE  IP          NODE
NOMINATED NODE  READINESS GATES
pod-s1    1/1    Running  0          20s  10.131.2.6  ci-ln-qg1il3k-f76d1-hlmhl-worker-b-df2s4
<none>    <none>

```

Pod に異なるノードセクターが含まれる場合、プロジェクトの Pod は作成またはスケジュールされません。たとえば、以下の Pod をサンプルプロジェクトにデプロイする場合、これは作成されません。

無効なノードセクターを持つ Pod オブジェクトの例

```

apiVersion: v1
kind: Pod
metadata:
  name: west-region
#...
spec:
  nodeSelector:
    region: west
#...

```

4.6.2. ノードセレクターの使用による Pod 配置の制御

Pod でノードセレクターを使用し、ノードでラベルを使用して、Pod がスケジュールされる場所を制御できます。ノードセレクターを使用すると、Red Hat OpenShift Service on AWS は一致するラベルが含まれるノード上に Pod をスケジュールします。

ラベルをノード、コンピューティングマシンセット、またはマシン設定に追加します。コンピューティングマシンセットにラベルを追加すると、ノードまたはマシンが停止した場合に、新規ノードにそのラベルが追加されます。ノードまたはマシン設定に追加されるラベルは、ノードまたはマシンが停止すると維持されません。

ノードセレクターを既存 Pod に追加するには、ノードセレクターを ReplicaSet オブジェクト、DaemonSet オブジェクト、StatefulSet オブジェクト、Deployment オブジェクト、または DeploymentConfig オブジェクトなどの Pod の制御オブジェクトに追加します。制御オブジェクト下の既存 Pod は、一致するラベルを持つノードで再作成されます。新規 Pod を作成する場合、ノードセレクターを Pod 仕様に直接追加できます。Pod に制御オブジェクトがない場合は、Pod を削除し、Pod 仕様を編集して、Pod を再作成する必要があります。



注記

ノードセレクターを既存のスケジュールされている Pod に直接追加することはできません。

前提条件

ノードセレクターを既存 Pod に追加するには、Pod の制御オブジェクトを判別します。たとえば、router-default-66d5cf9464-m2g75 Pod は router-default-66d5cf9464 レプリカセットによって制

御されます。

```
$ oc describe pod router-default-66d5cf9464-7pwkc
```

出力例

```
kind: Pod
apiVersion: v1
metadata:
# ...
Name:      router-default-66d5cf9464-7pwkc
Namespace: openshift-ingress
# ...
Controlled By:  ReplicaSet/router-default-66d5cf9464
# ...
```

Web コンソールでは、Pod YAML の `ownerReferences` に制御オブジェクトをリスト表示します。

```
apiVersion: v1
kind: Pod
metadata:
  name: router-default-66d5cf9464-7pwkc
# ...
ownerReferences:
- apiVersion: apps/v1
  kind: ReplicaSet
  name: router-default-66d5cf9464
  uid: d81dd094-da26-11e9-a48a-128e7edf0312
  controller: true
  blockOwnerDeletion: true
# ...
```

手順

- 一致するノードセクターを Pod に追加します。
- ノードセクターを既存 Pod および新規 Pod に追加するには、ノードセクターを Pod の制御オブジェクトに追加します。

ラベルを含む ReplicaSet オブジェクトのサンプル

```
kind: ReplicaSet
apiVersion: apps/v1
metadata:
  name: hello-node-6fbccf8d9
# ...
spec:
# ...
  template:
    metadata:
      creationTimestamp: null
    labels:
      ingresscontroller.operator.openshift.io/deployment-ingresscontroller: default
      pod-template-hash: 66d5cf9464
    spec:
      nodeSelector:
        kubernetes.io/os: linux
        node-role.kubernetes.io/worker: ""
        type: user-node ❶
# ...
```

❶

ノードセレクターを追加します。

○

ノードセレクターを特定の新規 Pod に追加するには、セレクターを Pod オブジェクトに直接追加します。

ノードセレクターを持つ Pod オブジェクトの例

```
apiVersion: v1
kind: Pod
metadata:
  name: hello-node-6fbccf8d9
# ...
spec:
  nodeSelector:
    region: east
    type: user-node
# ...
```



注記

ノードセクターを既存のスケジュールされている Pod に直接追加することはできません。

関連情報

- [ノードセクターおよび容認を使用したプロジェクトの作成](#)

4.7. POD トポロジー分散制約を使用した POD 配置の制御

Pod トポロジー分散制約を使用して、ノード、ゾーン、リージョンその他のユーザー定義のトポロジードメイン間で Pod の配置を制御できます。

4.7.1. Pod トポロジー分散制約について

Pod トポロジー分散制約を使用することで、障害ドメイン全体にまたがる Pod の分散に対する詳細な制御を実現し、高可用性とより効率的なリソースの使用を実現できます。

Red Hat OpenShift Service on AWS 管理者はノードにラベルを付け、リージョン、ゾーン、ノード、他のユーザー定義ドメインなどのトポロジー情報を提供できます。これらのラベルをノードに設定した後に、ユーザーは Pod トポロジーの分散制約を定義し、これらのトポロジードメイン全体での Pod の配置を制御できます。

グループ化する Pod を指定し、それらの Pod が分散されるトポロジードメインと、許可できるスケジューリングを指定します。制約により、分散される際に同じ namespace 内の Pod のみが一致し、グループ化されます。

4.7.2. Pod トポロジー分散制約の設定

以下の手順は、Pod トポロジー分散制約を、ゾーンに基づいて指定されたラベルに一致する Pod を分散するように設定する方法を示しています。

複数の Pod トポロジー分散制約を指定できますが、それらが互いに競合しないようにする必要があります。Pod を配置するには、すべての Pod トポロジー分散制約を満たしている必要があります。

前提条件

- **dedicated-admin** ロールを持つユーザーが、必要なラベルをノードに追加している。

手順

1. Pod 仕様を作成し、Pod トポロジーの分散制約を指定します。

pod-spec.yaml ファイルの例

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
  labels:
    region: us-east
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  topologySpreadConstraints:
    - maxSkew: 1 ①
      topologyKey: topology.kubernetes.io/zone ②
      whenUnsatisfiable: DoNotSchedule ③
      labelSelector: ④
        matchLabels:
          region: us-east ⑤
      matchLabelKeys:
        - my-pod-label ⑥
  containers:
    - image: "docker.io/ocpqe/hello-pod"
      name: hello-pod
      securityContext:
        allowPrivilegeEscalation: false
      capabilities:
        drop: [ALL]
```

①

任意の 2 つのトポロジードメイン間の Pod 数の最大差。デフォルトは 1 で、0 の値を指定することはできません。

2

ノードラベルのキー。このキーと同じ値を持つノードは同じトポロジーにあると見なされます。

3

分散制約を満たさない場合に Pod を処理する方法です。デフォルトは `DoNotSchedule` であり、これはスケジューラーに Pod をスケジューリングしないように指示します。 `ScheduleAnyway` に設定して Pod を依然としてスケジューリングできますが、スケジューラーはクラスターがさらに不均衡な状態になるのを防ぐためにスキューの適用を優先します。

4

制約を満たすために、分散される際に、このラベルセレクターに一致する Pod はグループとしてカウントされ、認識されます。ラベルセレクターを指定してください。指定しないと、Pod が一致しません。

5

今後適切にカウントされるようにするには、この Pod 仕様がこのラベルセレクターに一致するようにラベルを設定していることも確認してください。

6

拡散を計算する Pod を選択するための Pod ラベルキーのリスト。

2.

Pod を作成します。

```
$ oc create -f pod-spec.yaml
```

4.7.3. Pod トポロジー分散制約の例

以下の例は、Pod トポロジー設定分散制約の設定を示しています。

4.7.3.1. 単一 Pod トポロジー分散制約の例

このサンプル Pod 仕様は単一の Pod トポロジー分散制約を定義します。これは `region: us-east` というラベルが付いた Pod で一致し、ゾーン間で分散され、スキューの 1 を指定し、これらの要件を満たさない場合に Pod をスケジューリングしません。

```
kind: Pod
apiVersion: v1
metadata:
  name: my-pod
  labels:
    region: us-east
spec:
  securityContext:
    runAsNonRoot: true
  seccompProfile:
    type: RuntimeDefault
  topologySpreadConstraints:
  - maxSkew: 1
    topologyKey: topology.kubernetes.io/zone
    whenUnsatisfiable: DoNotSchedule
  labelSelector:
    matchLabels:
      region: us-east
  containers:
  - image: "docker.io/ocpqe/hello-pod"
    name: hello-pod
    securityContext:
      allowPrivilegeEscalation: false
    capabilities:
      drop: [ALL]
```

4.7.3.2. 複数の Pod トポロジー分散制約の例

このサンプル Pod 仕様は 2 つの Pod トポロジー分散制約を定義します。どちらも `region: us-east` というラベルの付いた Pod に一致し、スキューを 1 に指定し、これらの要件を満たしていない場合は Pod はスケジュールされません。

最初の制約は、ユーザー定義ラベルの `node` に基づいて Pod を分散し、2 つ目の制約はユーザー定義ラベルの `rack` に基づいて Pod を分散します。Pod がスケジュールされるには、両方の制約を満たす必要があります。

```
kind: Pod
apiVersion: v1
metadata:
  name: my-pod-2
  labels:
    region: us-east
spec:
  securityContext:
    runAsNonRoot: true
  seccompProfile:
    type: RuntimeDefault
  topologySpreadConstraints:
  - maxSkew: 1
    topologyKey: node
    whenUnsatisfiable: DoNotSchedule
```

```
labelSelector:  
  matchLabels:  
    region: us-east  
- maxSkew: 1  
  topologyKey: rack  
  whenUnsatisfiable: DoNotSchedule  
labelSelector:  
  matchLabels:  
    region: us-east  
containers:  
- image: "docker.io/ocpqe/hello-pod"  
  name: hello-pod  
  securityContext:  
    allowPrivilegeEscalation: false  
  capabilities:  
    drop: [ALL]
```

第5章 ジョブと DEAMONSET の使用

5.1. デーモンセットによるノード上でのバックグラウンドタスクの自動的な実行

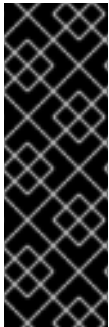
管理者は、デーモンセットを作成して使用し、Red Hat OpenShift Service on AWS クラスタ内の特定のノードまたはすべてのノードで Pod のレプリカを実行できます。

デーモンセットは、すべて (または一部) のノードで Pod のコピーが確実に実行されるようにします。ノードがクラスタに追加されると、Pod がクラスタに追加されます。ノードがクラスタから削除されると、Pod はガベージコレクションによって削除されます。デーモンセットを削除すると、デーモンセットによって作成された Pod がクリーンアップされます。

デーモンセットを使用して共有ストレージを作成し、クラスタ内のすべてのノードでロギング Pod を実行するか、すべてのノードでモニターエージェントをデプロイできます。

セキュリティ上の理由から、クラスタ管理者とプロジェクト管理者がデーモンセットを作成できません。

デーモンセットについての詳細は、[Kubernetes ドキュメント](#) を参照してください。



重要

デーモンセットのスケジューリングにはプロジェクトのデフォルトノードセレクターとの互換性がありません。これを無効にしない場合、デーモンセットはデフォルトのノードセレクターとのマージによって制限されます。これにより、マージされたノードセレクターで選択解除されたノードで Pod が頻繁に再作成されるようになり、クラスタに不要な負荷が加わります。

5.1.1. デフォルトスケジューラーによるスケジュール

デーモンセットは、適格なすべてのノードで Pod のコピーが確実に実行されるようにします。通常は、Pod が実行されるノードは Kubernetes のスケジューラーが選択します。ただし、デーモンセット Pod はデーモンセットコントローラーによって作成され、スケジュールされます。その結果、以下のような問題が生じています。

- Pod の動作に一貫性がない。スケジューリングを待機している通常の Pod は、作成されると Pending 状態になりますが、デーモンセット Pod は作成されても Pending 状態になりません。これによりユーザーに混乱が生じます。

- Pod のプリエンプションがデフォルトのスケジューラーで処理される。プリエンプションが有効にされると、デーモンセットコントローラーは Pod の優先順位とプリエンプションを考慮することなくスケジューリングの決定を行います。

ScheduleDaemonSetPods 機能は、Red Hat OpenShift Service on AWS でデフォルトで有効にされます。これにより、spec.nodeName の条件 (term) ではなく NodeAffinity の条件 (term) をデーモンセット Pod に追加することで、デーモンセットコントローラーではなくデフォルトのスケジューラーを使ってデーモンセットをスケジュールすることができます。その後、デフォルトのスケジューラーは、Pod をターゲットホストにバインドさせるために使用されます。デーモンセット Pod のノードアフィニティーがすでに存在する場合、これは置き換えられます。デーモンセットコントローラーは、デーモンセット Pod を作成または変更する場合にのみこれらの操作を実行し、デーモンセットの spec.template は一切変更されません。

```
kind: Pod
apiVersion: v1
metadata:
  name: hello-node-6fbccf8d9-9tmzr
#...
spec:
  nodeAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
      - matchFields:
        - key: metadata.name
          operator: In
          values:
            - target-host-name
#...
```

さらに、node.kubernetes.io/unschedulable:NoSchedule の容認がデーモンセット Pod に自動的に追加されます。デフォルトのスケジューラーは、デーモンセット Pod をスケジュールする際に、スケジュールできないノードを無視します。

5.1.2. デーモンセットの作成

デーモンセットの作成時に、nodeSelector フィールドは、デーモンセットがレプリカをデプロイする必要のあるノードを指定するために使用されます。

前提条件

- デーモンセットの使用を開始する前に、namespace のアノテーション openshift.io/node-selector を空の文字列に設定することで、namespace のプロジェクトスコープのデフォルトのノードセレクターを無効にします。

```
$ oc patch namespace myproject -p \
  '{"metadata": {"annotations": {"openshift.io/node-selector": ""}}}'
```

ヒント

または、以下の YAML を適用して、プロジェクト全体で namespace のデフォルトのノードセレクターを無効にすることもできます。

```
apiVersion: v1
kind: Namespace
metadata:
  name: <namespace>
  annotations:
    openshift.io/node-selector: "
#...
```

手順

デーモンセットを作成するには、以下を実行します。

1. デーモンセット yaml ファイルを定義します。

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: hello-daemonset
spec:
  selector:
    matchLabels:
      name: hello-daemonset ①
  template:
    metadata:
      labels:
        name: hello-daemonset ②
    spec:
      nodeSelector: ③
        role: worker
      containers:
        - image: openshift/hello-openshift
          imagePullPolicy: Always
          name: registry
          ports:
            - containerPort: 80
              protocol: TCP
          resources: {}
          terminationMessagePath: /dev/termination-log
```

```
serviceAccount: default
terminationGracePeriodSeconds: 10
#...
```

1

デーモンセットに属する Pod を判別するラベルセクターです。

2

Pod テンプレートのラベルセクターです。上記のラベルセクターに一致している必要があります。

3

Pod レプリカをデプロイする必要があるノードを判別するノードセクターです。一致するラベルがこのノードに存在する必要があります。

2.

デーモンセットオブジェクトを作成します。

```
$ oc create -f daemonset.yaml
```

3.

Pod が作成されていることを確認し、各 Pod に Pod レプリカがあることを確認するには、以下を実行します。

a.

daemonset Pod を検索します。

```
$ oc get pods
```

出力例

```
hello-daemonset-cx6md 1/1    Running 0    2m
hello-daemonset-e3md9 1/1    Running 0    2m
```

b.

Pod がノードに配置されていることを確認するために Pod を表示します。

```
$ oc describe pod/hello-daemonset-cx6md|grep Node
```

出力例

```
Node:      openshift-node01.hostname.com/10.14.20.134
```

```
$ oc describe pod/hello-daemonset-e3md9|grep Node
```

出力例

```
Node:      openshift-node02.hostname.com/10.14.20.137
```

重要

- デーモンセット Pod テンプレートを更新しても、既存の Pod レプリカには影響はありません。
- デーモンセットを削除してから、異なるテンプレートと同じラベルセクターを使用して新規のデーモンセットを作成する場合に、既存の Pod レプリカについてラベルが一致していると認識するため、既存の Pod レプリカは更新されず、Pod テンプレートで一致しない場合でも新しいレプリカが作成されます。
- ノードのラベルを変更する場合には、デーモンセットは新しいラベルと一致するノードに Pod を追加し、新しいラベルと一致しないノードから Pod を削除します。

デーモンセットを更新するには、古いレプリカまたはノードを削除して新規の Pod レプリカの作成を強制的に実行します。

5.2. ジョブを使用した POD でのタスクの実行

ジョブは、Red Hat OpenShift Service on AWS クラスタでタスクを実行します。

ジョブは、タスクの全体的な進捗状況を追跡し、進行中、完了、および失敗した各 Pod の情報を使用してその状態を更新します。ジョブを削除するとそのジョブによって作成された Pod のレプリカがクリーンアップされます。ジョブは Kubernetes API の一部で、他のオブジェクトタイプ同様に oc コマンドで管理できます。

ジョブ仕様のサンプル

```
apiVersion: batch/v1
kind: Job
metadata:
  name: pi
spec:
  parallelism: 1 ①
  completions: 1 ②
  activeDeadlineSeconds: 1800 ③
  backoffLimit: 6 ④
  template: ⑤
    metadata:
      name: pi
    spec:
      containers:
        - name: pi
          image: perl
          command: ["perl", "-Mbignum=bpi", "-wle", "print bpi(2000)"]
      restartPolicy: OnFailure ⑥
#...
```

①

ジョブの Pod レプリカは並行して実行される必要があります。

②

ジョブの完了をマークするには、Pod の正常な完了が必要です。

③

ジョブを実行できる最長期間。

4

ジョブの再試行回数。

5

コントローラーが作成する Pod のテンプレート。

6

Pod の再起動ポリシー。

関連情報

- [Kubernetes ドキュメントの ジョブ](#)

5.2.1. ジョブと Cron ジョブについて

ジョブは、タスクの全体的な進捗状況を追跡し、進行中、完了、および失敗した各 Pod の情報を使用してその状態を更新します。ジョブを削除するとそのジョブによって作成された Pod がクリーンアップされます。ジョブは Kubernetes API の一部で、他のオブジェクトタイプ同様に oc コマンドで管理できます。

Red Hat OpenShift Service on AWS で一度だけ実行するオブジェクトを作成できるリソースタイプは 2 種類あります。

ジョブ

定期的なジョブは、タスクを作成しジョブが完了したことを確認する、一度だけ実行するオブジェクトです。

ジョブとして実行するには、主に以下のタスクタイプを使用できます。

- **非並列ジョブ:**
 - Pod が失敗しない限り、単一の Pod のみを起動するジョブ。
 - このジョブは、Pod が正常に終了するとすぐに完了します。

- 固定の完了数が指定された並列ジョブ
 - 複数の Pod を起動するジョブ。
 - ジョブはタスク全体を表し、1 から completions 値までの範囲内のそれぞれの値に対して 1 つの正常な Pod がある場合に完了します。
 - ワークキューを含む並列ジョブ:
 - 指定された Pod に複数の並列ワーカークラスを持つジョブ。
 - Red Hat OpenShift Service on AWS は、Pod を調整して各 Pod が何を処理するかを決定するか、外部キューサービスを使用します。
 - 各 Pod はそれぞれ、すべてのピア Pod が完了しているかどうかや、ジョブ全体が実行済みであることを判別することができます。
 - ジョブからの Pod が正常な状態で終了すると、新規 Pod は作成されません。
 - 1 つ以上の Pod が正常な状態で終了し、すべての Pod が終了している場合、ジョブが正常に完了します。
 - Pod が正常な状態で終了した場合、それ以外の Pod がこのタスクについて機能したり、または出力を書き込むことはありません。Pod はすべて終了プロセスにあるはずです。
- 各種のジョブを使用する方法についての詳細は、Kubernetes ドキュメントの [Job Patterns](#) を参照してください。

Cron ジョブ

ジョブは、Cron ジョブを使用して複数回実行するようにスケジューリングすることが可能です。

cron ジョブは、ユーザーがジョブの実行方法を指定することを可能にすることで、定期的な

ジョブを積み重ねます。Cron ジョブは **Kubernetes** API の一部であり、他のオブジェクトタイプと同様に `oc` コマンドで管理できます。

Cron ジョブは、バックアップの実行やメールの送信など周期的な繰り返しのタスクを作成する際に役立ちます。また、低アクティビティー期間にジョブをスケジュールする場合など、特定の時間に個別のタスクをスケジュールすることも可能です。cron ジョブは、cronjob コントローラーを実行するコントロールプレーンノードに設定されたタイムゾーンに基づいて Job オブジェクトを作成します。



警告

Cron ジョブはスケジュールの実行時間ごとに約 1 回ずつ Job オブジェクトを作成しますが、ジョブの作成に失敗したり、2つのジョブが作成される場合があります。そのためジョブはべき等である必要があり、履歴制限を設定する必要があります。

5.2.1.1. ジョブの作成方法

どちらのリソースタイプにも、以下の主要な要素から構成されるジョブ設定が必要です。

- Red Hat OpenShift Service on AWS が作成する Pod を記述している Pod テンプレート。
- `parallelism` パラメーター。ジョブの実行に使用する、同時に実行される Pod の数を指定します。
 - 非並列ジョブの場合は、未設定のままにします。未設定の場合は、デフォルトの 1 に設定されます。
- `completions` パラメーター。ジョブを完了するために必要な、正常に完了した Pod の数を指定します。
 - 非並列ジョブの場合は、未設定のままにします。未設定の場合は、デフォルトの 1 に設定されます。

- 固定の完了数を持つ並列ジョブの場合は、値を指定します。
- ワークキューのある並列ジョブでは、未設定のままにします。未設定の場合、デフォルトは `parallelism` 値に設定されます。

5.2.1.2. ジョブの最長期間を設定する方法

ジョブの定義時に、`activeDeadlineSeconds` フィールドを設定して最長期間を定義できます。これは秒単位で指定され、デフォルトでは設定されません。設定されていない場合は、実施される最長期間はありません。

最長期間は、最初の Pod がスケジュールされた時点から計算され、ジョブが有効である期間を定義します。これは実行の全体の時間を追跡します。指定されたタイムアウトに達すると、Red Hat OpenShift Service on AWS がジョブを終了します。

5.2.1.3. 失敗した Pod のためのジョブのバックオフポリシーを設定する方法

ジョブは、設定の論理的なエラーなどの理由により再試行の設定回数を超えた後に失敗とみなされる場合があります。ジョブに関連付けられた失敗した Pod は 6 分を上限として指数関数的バックオフ遅延値 (10s、20s、40s ...) に基づいて再作成されます。この制限は、コントローラーのチェック間で失敗した Pod が新たに生じない場合に再設定されます。

ジョブの再試行回数を設定するには `spec.backoffLimit` パラメーターを使用します。

5.2.1.4. アーティファクトを削除するように Cron ジョブを設定する方法

Cron ジョブはジョブや Pod などのアーティファクトリソースをそのままにすることがあります。ユーザーは履歴制限を設定して古いジョブとそれらの Pod が適切に消去されるようにすることが重要です。これに対応する 2 つのフィールドが Cron ジョブ仕様にあります。

- `.spec.successfulJobsHistoryLimit`. 保持する成功した終了済みジョブの数 (デフォルトは 3 に設定)。
- `.spec.failedJobsHistoryLimit`. 保持する失敗した終了済みジョブの数 (デフォルトは 1 に設定)。

5.2.1.5. 既知の制限

ジョブ仕様の再起動ポリシーは Pod にのみ適用され、ジョブコントローラーには適用されません。ただし、ジョブコントローラーはジョブを完了まで再試行するようハードコーディングされます。

そのため `restartPolicy: Never` または `--restart=Never` により、`restartPolicy: OnFailure` または `--restart=OnFailure` と同じ動作が実行されます。つまり、ジョブが失敗すると、成功するまで (または手動で破棄されるまで) 自動で再起動します。このポリシーは再起動するサブシステムのみを設定します。

`Never` ポリシーでは、ジョブコントローラーが再起動を実行します。それぞれの再試行時に、ジョブコントローラーはジョブステータスの失敗数を増分し、新規 Pod を作成します。これは、それぞれの試行が失敗するたびに Pod の数が増えることを意味します。

`OnFailure` ポリシーでは、`kubelet` が再起動を実行します。それぞれの試行によりジョブステータスでの失敗数が増分する訳ではありません。さらに、`kubelet` は同じノードで Pod の起動に失敗したジョブを再試行します。

5.2.2. ジョブの作成

Red Hat OpenShift Service on AWS でジョブを作成するには、ジョブオブジェクトを作成します。

手順

ジョブを作成するには、以下を実行します。

1. 以下のような YAML ファイルを作成します。

```
apiVersion: batch/v1
kind: Job
metadata:
  name: pi
spec:
  parallelism: 1 ①
  completions: 1 ②
  activeDeadlineSeconds: 1800 ③
  backoffLimit: 6 ④
  template: ⑤
    metadata:
      name: pi
```

```
spec:
  containers:
    - name: pi
      image: perl
      command: ["perl", "-Mbignum=bpi", "-wle", "print bpi(2000)"]
      restartPolicy: OnFailure ⑥
#...
```

①

オプション: ジョブを並行して実行する Pod レプリカの数指定します。デフォルトは 1 です。

- 非並列ジョブの場合は、未設定のままにします。未設定の場合は、デフォルトの 1 に設定されます。

②

オプション: ジョブの完了をマークするために必要な Pod の正常な完了の数を指定します。

- 非並列ジョブの場合は、未設定のままにします。未設定の場合は、デフォルトの 1 に設定されます。
- 固定の完了数を持つ並列ジョブの場合、完了の数を指定します。
- ワークキューのある並列ジョブでは、未設定のままにします。未設定の場合、デフォルトは `parallelism` 値に設定されます。

③

オプション: ジョブを実行できる最大期間を指定します。

④

オプション: ジョブの再試行回数を指定します。このフィールドは、デフォルトでは 6 に設定されています。

⑤

コントローラーが作成する Pod のテンプレートを指定します。

⑥

Pod の再起動ポリシーを指定します。

- Never. ジョブを再起動しません。
- OnFailure. ジョブが失敗した場合にのみ再起動します。
- Always. ジョブを常に再起動します。

Red Hat OpenShift Service on AWS が失敗したコンテナに再起動ポリシーを使用する方法の詳細は、Kubernetes ドキュメントの [Example States](#) を参照してください。

2. ジョブを作成します。

```
$ oc create -f <file-name>.yaml
```

注記

`oc create job` を使用して単一コマンドからジョブを作成し、起動することもできます。以下のコマンドは直前の例に指定されている同じジョブを作成し、これを起動します。

```
$ oc create job pi --image=perl -- perl -Mbignum=bpi -wle 'print bpi(2000)'
```

5.2.3. cron ジョブの作成

Red Hat OpenShift Service on AWS で cron ジョブを作成するには、ジョブオブジェクトを作成します。

手順

cron ジョブを作成するには、以下を実行します。

1. 以下のような YAML ファイルを作成します。

```

apiVersion: batch/v1
kind: CronJob
metadata:
  name: pi
spec:
  schedule: "*/1 * * * *" ①
  concurrencyPolicy: "Replace" ②
  startingDeadlineSeconds: 200 ③
  suspend: true ④
  successfulJobsHistoryLimit: 3 ⑤
  failedJobsHistoryLimit: 1 ⑥
  jobTemplate: ⑦
    spec:
      template:
        metadata:
          labels: ⑧
            parent: "cronjobpi"
        spec:
          containers:
            - name: pi
              image: perl
              command: ["perl", "-Mbignum=bpi", "-wle", "print bpi(2000)"]
          restartPolicy: OnFailure ⑨

```

①

cron 形式 で指定されたジョブのスケジュール。この例では、ジョブは毎分実行されます。

②

オプションの同時実行ポリシー。cron ジョブ内での同時実行ジョブを処理する方法を指定します。以下の同時実行ポリシーの 1 つのみを指定できます。これが指定されない場合、同時実行を許可するようにデフォルト設定されます。

- **Allow:** Cron ジョブを同時に実行できます。
- **Forbid:** 同時実行を禁止し、直前の実行が終了していない場合は次の実行を省略します。
- **Replace:** 同時に実行されているジョブを取り消し、これを新規ジョブに置き換えます。

③

ジョブを開始するためのオプションの期限 (秒単位)(何らかの理由によりスケジュールされた時間が経過する場合)。ジョブの実行が行われない場合、ジョブの失敗としてカウ

ントされます。これが指定されない場合は期間が設定されません。

4

Cron ジョブの停止を許可するオプションのフラグ。これが `true` に設定されている場合、後続のすべての実行が停止されます。

5

保持する成功した終了済みジョブの数 (デフォルトは 3 に設定)。

6

保持する失敗した終了済みジョブの数 (デフォルトは 1 に設定)。

7

ジョブテンプレート。これはジョブの例と同様です。

8

この Cron ジョブで生成されるジョブのラベルを設定します。

9

Pod の再起動ポリシー。ジョブコントローラーには適用されません。



注記

`.spec.successfulJobsHistoryLimit` と `.spec.failedJobsHistoryLimit` のフィールドはオプションです。これらのフィールドでは、完了したジョブと失敗したジョブのそれぞれを保存する数を指定します。デフォルトで、これらのジョブの保存数はそれぞれ 3 と 1 に設定されます。制限に 0 を設定すると、終了後に対応する種類のジョブのいずれも保持しません。

2.

`cron` ジョブを作成します。

```
$ oc create -f <file-name>.yaml
```



注記

`oc create cronjob` を使用して単一コマンドから cron ジョブを作成し、起動することもできます。以下のコマンドは直前の例で指定されている同じ cron ジョブを作成し、これを起動します。

```
$ oc create cronjob pi --image=perl --schedule='*/1 * * * *' -- perl -Mbignum=bpi -wle 'print bpi(2000)'
```

`oc create cronjob` で、`--schedule` オプションは **cron 形式** のスケジュールを受け入れます。

第6章 ノードの使用

6.1. RED HAT OPENSIFT SERVICE ON AWS クラスタ内のノードの閲覧とリスト表示

クラスタのすべてのノードをリスト表示し、ステータスや経過時間、メモリー使用量などの情報およびノードについての詳細を取得できます。

ノード管理の操作を実行すると、CLI は実際のノードホストの表現であるノードオブジェクトと対話します。マスターはノードオブジェクトの情報を使用してヘルスチェックでノードを検証します。

6.1.1. クラスタ内のすべてのノードのリスト表示について

クラスタ内のノードに関する詳細な情報を取得できます。

- 以下のコマンドは、すべてのノードをリスト表示します。

```
$ oc get nodes
```

以下の例は、正常なノードを持つクラスタです。

```
$ oc get nodes
```

出力例

```
NAME                STATUS  ROLES  AGE  VERSION
master.example.com  Ready  master  7h   v1.28.5
node1.example.com   Ready  worker  7h   v1.28.5
node2.example.com   Ready  worker  7h   v1.28.5
```

以下の例は、正常でないノードが 1 つ含まれるクラスタです。

```
$ oc get nodes
```

出力例

| NAME | STATUS | ROLES | AGE | VERSION | |
|--------------------|-----------------------------|--------|-----|---------|---------|
| master.example.com | Ready | master | 7h | v1.28.5 | |
| node1.example.com | NotReady,SchedulingDisabled | worker | 7h | v1.28.5 | v1.28.5 |
| node2.example.com | Ready | worker | 7h | v1.28.5 | |

NotReady ステータスをトリガーする条件については、本セクションの後半で説明します。

- -o wide オプションは、ノードについての追加情報を提供します。

```
$ oc get nodes -o wide
```

出力例

| NAME | STATUS | ROLES | AGE | VERSION | INTERNAL-IP | EXTERNAL-IP | OS-IMAGE | KERNEL-VERSION | CONTAINER-RUNTIME |
|--------------------|--------|--------|------|---------|--------------|-------------|--|------------------------------|--|
| master.example.com | Ready | master | 171m | v1.28.5 | 10.0.129.108 | <none> | Red Hat Enterprise Linux CoreOS 48.83.202103210901-0 (Ootpa) | 4.18.0-240.15.1.el8_3.x86_64 | cri-o://1.28.5-30.rhaos4.10.gitf2f339d.el8-dev |
| node1.example.com | Ready | worker | 72m | v1.28.5 | 10.0.129.222 | <none> | Red Hat Enterprise Linux CoreOS 48.83.202103210901-0 (Ootpa) | 4.18.0-240.15.1.el8_3.x86_64 | cri-o://1.28.5-30.rhaos4.10.gitf2f339d.el8-dev |
| node2.example.com | Ready | worker | 164m | v1.28.5 | 10.0.142.150 | <none> | Red Hat Enterprise Linux CoreOS 48.83.202103210901-0 (Ootpa) | 4.18.0-240.15.1.el8_3.x86_64 | cri-o://1.28.5-30.rhaos4.10.gitf2f339d.el8-dev |

- 以下のコマンドは、単一のノードに関する情報をリスト表示します。

```
$ oc get node <node>
```

以下に例を示します。

```
$ oc get node node1.example.com
```

出力例

```
NAME                STATUS  ROLES  AGE   VERSION
node1.example.com   Ready  worker  7h   v1.28.5
```

- 以下のコマンドを実行すると、現在の状態の理由を含む、特定ノードについての詳細情報を取得できます。

```
$ oc describe node <node>
```

以下に例を示します。

```
$ oc describe node node1.example.com
```

出力例

```
Name:                node1.example.com ①
Roles:               worker ②
Labels:              kubernetes.io/os=linux
                    kubernetes.io/hostname=ip-10-0-131-14
                    kubernetes.io/arch=amd64 ③
                    node-role.kubernetes.io/worker=
                    node.kubernetes.io/instance-type=m4.large
                    node.openshift.io/os_id=rhcos
                    node.openshift.io/os_version=4.5
                    region=east
                    topology.kubernetes.io/region=us-east-1
                    topology.kubernetes.io/zone=us-east-1a
Annotations:        cluster.k8s.io/machine: openshift-machine-api/ahardin-worker-us-
east-2a-q5dzc ④
                    machineconfiguration.openshift.io/currentConfig: worker-
309c228e8b3a92e2235edd544c62fea8
                    machineconfiguration.openshift.io/desiredConfig: worker-
309c228e8b3a92e2235edd544c62fea8
                    machineconfiguration.openshift.io/state: Done
                    volumes.kubernetes.io/controller-managed-attach-detach: true
CreationTimestamp:  Wed, 13 Feb 2019 11:05:57 -0500
Taints:              <none> ⑤
```

```

Unschedulable: false
Conditions: 6
  Type          Status LastHeartbeatTime          LastTransitionTime          Reason
  Message
  ----          -
  OutOfDisk     False  Wed, 13 Feb 2019 15:09:42 -0500  Wed, 13 Feb 2019 11:05:57 -
0500 KubeletHasSufficientDisk  kubelet has sufficient disk space available
  MemoryPressure False  Wed, 13 Feb 2019 15:09:42 -0500  Wed, 13 Feb 2019
11:05:57 -0500 KubeletHasSufficientMemory  kubelet has sufficient memory
available
  DiskPressure  False  Wed, 13 Feb 2019 15:09:42 -0500  Wed, 13 Feb 2019 11:05:57 -
0500 KubeletHasNoDiskPressure  kubelet has no disk pressure
  PIDPressure   False  Wed, 13 Feb 2019 15:09:42 -0500  Wed, 13 Feb 2019 11:05:57 -
0500 KubeletHasSufficientPID    kubelet has sufficient PID available
  Ready         True   Wed, 13 Feb 2019 15:09:42 -0500  Wed, 13 Feb 2019 11:07:09 -
0500 KubeletReady              kubelet is posting ready status
Addresses: 7
  InternalIP: 10.0.140.16
  InternalDNS: ip-10-0-140-16.us-east-2.compute.internal
  Hostname: ip-10-0-140-16.us-east-2.compute.internal
Capacity: 8
  attachable-volumes-aws-ebs: 39
  cpu: 2
  hugepages-1Gi: 0
  hugepages-2Mi: 0
  memory: 8172516Ki
  pods: 250
Allocatable:
  attachable-volumes-aws-ebs: 39
  cpu: 1500m
  hugepages-1Gi: 0
  hugepages-2Mi: 0
  memory: 7558116Ki
  pods: 250
System Info: 9
  Machine ID: 63787c9534c24fde9a0cde35c13f1f66
  System UUID: EC22BF97-A006-4A58-6AF8-0A38DEEA122A
  Boot ID: f24ad37d-2594-46b4-8830-7f7555918325
  Kernel Version: 3.10.0-957.5.1.el7.x86_64
  OS Image: Red Hat Enterprise Linux CoreOS 410.8.20190520.0
(Ootpa)
  Operating System: linux
  Architecture: amd64
  Container Runtime Version: cri-o://1.28.5-0.6.dev.rhaos4.3.git9ad059b.el8-rc2
  Kubelet Version: v1.28.5
  Kube-Proxy Version: v1.28.5
  PodCIDR: 10.128.4.0/24
  ProviderID: aws:///us-east-2a/i-04e87b31dc6b3e171
  Non-terminated Pods: (12 in total) 10
  Namespace          Name          CPU Requests  CPU Limits
  Memory Requests  Memory Limits
  -----
  openshift-cluster-node-tuning-operator tuned-hdl5q  0 (0%)  0 (0%)
0 (0%)  0 (0%)

```

```

openshift-dns                dns-default-l69zr           0 (0%)    0 (0%)    0 (0%)
0 (0%)
openshift-image-registry     node-ca-9hmcg               0 (0%)    0 (0%)    0
(0%)    0 (0%)
openshift-ingress            router-default-76455c45c-c5ptv  0 (0%)    0 (0%)
0 (0%)    0 (0%)
openshift-machine-config-operator  machine-config-daemon-cvqw9    20m
(1%)    0 (0%)    50Mi (0%)    0 (0%)
openshift-marketplace        community-operators-f67fh      0 (0%)    0 (0%)
0 (0%)    0 (0%)
openshift-monitoring         alertmanager-main-0           50m (3%)   50m
(3%)    210Mi (2%)    10Mi (0%)
openshift-monitoring         node-exporter-l7q8d           10m (0%)   20m
(1%)    20Mi (0%)    40Mi (0%)
openshift-monitoring         prometheus-adapter-75d769c874-hvb85  0 (0%)
0 (0%)    0 (0%)    0 (0%)
openshift-multus             multus-kw8w5                  0 (0%)    0 (0%)    0
(0%)    0 (0%)
openshift-sdn                ovs-t4dsn                     100m (6%)  0 (0%)    300Mi
(4%)    0 (0%)
openshift-sdn                sdn-g79hg                     100m (6%)  0 (0%)
200Mi (2%)    0 (0%)
Allocated resources:
(Total limits may be over 100 percent, i.e., overcommitted.)
Resource           Requests  Limits
-----
cpu                 380m (25%)  270m (18%)
memory              880Mi (11%)  250Mi (3%)
attachable-volumes-aws-efs 0           0
Events: 11
Type Reason Age From Message
----
Normal NodeHasSufficientPID 6d (x5 over 6d) kubelet, m01.example.com Node
m01.example.com status is now: NodeHasSufficientPID
Normal NodeAllocatableEnforced 6d kubelet, m01.example.com Updated
Node Allocatable limit across pods
Normal NodeHasSufficientMemory 6d (x6 over 6d) kubelet, m01.example.com
Node m01.example.com status is now: NodeHasSufficientMemory
Normal NodeHasNoDiskPressure 6d (x6 over 6d) kubelet, m01.example.com
Node m01.example.com status is now: NodeHasNoDiskPressure
Normal NodeHasSufficientDisk 6d (x6 over 6d) kubelet, m01.example.com Node
m01.example.com status is now: NodeHasSufficientDisk
Normal NodeHasSufficientPID 6d kubelet, m01.example.com Node
m01.example.com status is now: NodeHasSufficientPID
Normal Starting 6d kubelet, m01.example.com Starting kubelet.
#...

```

1

ノードの名前。

2

ノードのロール (master または worker のいずれか)。

3

ノードに適用されたラベル。

4

ノードに適用されるアノテーション。

5

ノードに適用されたテイント。

6

ノードの状態およびステータス。conditions スタンザは、Ready、PIDPressure、PIDPressure、MemoryPressure、DiskPressure および OutOfDisk ステータスをリスト表示します。これらの状態については、本セクションの後半で説明します。

7

ノードの IP アドレスとホスト名。

8

Pod のリソースと割り当て可能なリソース。

9

ノードホストについての情報。

10

ノードの Pod。

11

ノードが報告したイベント。

ノードについての情報の中でも、とりわけ以下のノードの状態がこのセクションで説明されるコマンドの出力に表示されます。

表6.1 ノードの状態

| 条件 | 説明 |
|---------------------------|--|
| Ready | true の場合、ノードは正常であり、Pod を受け入れることのできる準備状態にあります。 false の場合、ノードは正常ではなく、Pod を受け入れません。 unknown の場合、ノードコントローラーは node-monitor-grace-period (デフォルトは 40 秒) の間にハートビートをノードから受信しませんでした。 |
| DiskPressure | true の場合、ディスク容量は低くなります。 |
| MemoryPressure | true の場合、ノードのメモリーは低くなります。 |
| PIDPressure | true の場合、ノードのプロセスが多すぎます。 |
| OutOfDisk | true の場合、ノードには新しい Pod を追加するためのノード上の空きスペースが十分にありません。 |
| NetworkUnavailable | true の場合、ノードのネットワークは正しく設定されていません。 |
| NotReady | true の場合、コンテナのランタイムやネットワークなど基本のコンポーネントのいずれかに問題が発生しているか、それらがまだ設定されていません。 |
| SchedulingDisabled | ノードに配置するように Pod をスケジュールすることができません。 |

6.1.2. クラスターでのノード上の Pod のリスト表示

特定のノード上のすべての Pod をリスト表示できます。

手順

- 1 つ以上のノードにすべてまたは選択した Pod をリスト表示するには、以下を実行します。

```
$ oc describe node <node1> <node2>
```

以下に例を示します。

```
$ oc describe node ip-10-0-128-218.ec2.internal
```

- 選択したノードのすべてまたは選択した Pod をリスト表示するには、以下を実行します。

```
$ oc describe node --selector=<node_selector>
```

```
$ oc describe node --selector=kubernetes.io/os
```

または、以下を実行します。

```
$ oc describe node -l=<pod_selector>
```

```
$ oc describe node -l node-role.kubernetes.io/worker
```

- 終了した Pod を含む、特定のノード上のすべての Pod をリスト表示するには、以下を実行します。

```
$ oc get pod --all-namespaces --field-selector=spec.nodeName=<nodename>
```

6.1.3. ノードのメモリーと CPU 使用統計の表示

コンテナのランタイム環境を提供する、ノードについての使用状況の統計を表示できます。これらの使用状況の統計には CPU、メモリー、およびストレージの消費量が含まれます。

前提条件

- 使用状況の統計を表示するには、`cluster-reader` 権限が必要です。
- 使用状況の統計を表示するには、メトリクスをインストールする必要があります。

手順

- 使用状況の統計を表示するには、以下を実行します。

```
$ oc adm top nodes
```

出力例

| NAME | CPU(cores) | CPU% | MEMORY(bytes) | MEMORY% |
|--------------------------------------|------------|------|---------------|---------|
| ip-10-0-12-143.ec2.compute.internal | 1503m | 100% | 4533Mi | 61% |
| ip-10-0-132-16.ec2.compute.internal | 76m | 5% | 1391Mi | 18% |
| ip-10-0-140-137.ec2.compute.internal | 398m | 26% | 2473Mi | 33% |
| ip-10-0-142-44.ec2.compute.internal | 656m | 43% | 6119Mi | 82% |
| ip-10-0-146-165.ec2.compute.internal | 188m | 12% | 3367Mi | 45% |
| ip-10-0-19-62.ec2.compute.internal | 896m | 59% | 5754Mi | 77% |
| ip-10-0-44-193.ec2.compute.internal | 632m | 42% | 5349Mi | 72% |

- ラベルの付いたノードの使用状況の統計を表示するには、以下を実行します。

```
$ oc adm top node --selector="
```

フィルターに使用するセレクター (ラベルクエリー) を選択する必要があります。=、==、および != をサポートします。

6.2. NODE TUNING OPERATOR の使用

Node Tuning Operator について説明し、この **Operator** を使用し、**Tuned** デーモンのオーケストレーションを実行してノードレベルのチューニングを管理する方法について説明します。

目的

Node Tuning Operator は、**Tuned** デーモンを調整することでノードレベルのチューニングを管理し、パフォーマンスプロファイルコントローラーを使用して低レイテンシーのパフォーマンスを実現するのに役立ちます。ほとんどの高パフォーマンスアプリケーションでは、一定レベルのカーネルのチューニングが必要です。**Node Tuning Operator** は、ノードレベルの **sysctl** の統一された管理インターフェイスをユーザーに提供し、ユーザーが指定するカスタムチューニングを追加できるよう柔軟性を提供します。

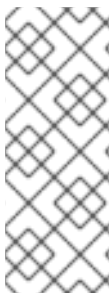
Operator は、コンテナ化された **Red Hat OpenShift Service on AWS** の **Tuned** デーモンを **Kubernetes** デーモンセットとして管理します。これにより、カスタムチューニング仕様が、デーモンが認識する形式でクラスターで実行されるすべてのコンテナ化された **Tuned** デーモンに渡されます。デーモンは、ノードごとに 1 つずつ、クラスターのすべてのノードで実行されます。

コンテナ化された **Tuned** デーモンによって適用されるノードレベルの設定は、プロファイルの変更をトリガーするイベントで、または終了シグナルの受信および処理によってコンテナ化された **Tuned** デーモンが正常に終了する際にロールバックされます。

Node Tuning Operator は、**Performance Profile** コントローラーを使用して自動チューニングを実装し、**Red Hat OpenShift Service on AWS** アプリケーションの低レイテンシーパフォーマンスを実現します。

クラスター管理者は、以下のようなノードレベルの設定を定義するパフォーマンスプロファイルを設定します。

- カーネルを `kernel-rt` に更新します。
- ハウスキーピング用の **CPU** を選択します。
- 実行中のワークロード用の **CPU** を選択します。



注記

現在、CPU 負荷分散の無効化は `cgroup v2` ではサポートされていません。その結果、`cgroup v2` が有効になっている場合は、パフォーマンスプロファイルから望ましい動作が得られない可能性があります。パフォーマンスプロファイルを使用している場合は、`cgroup v2` を有効にすることは推奨されません。

Node Tuning Operator は、バージョン 4.1 以降の標準の **Red Hat OpenShift Service on AWS** インストールに含まれています。



注記

Red Hat OpenShift Service on AWS の以前のバージョンでは、**OpenShift** アプリケーションの低レイテンシーパフォーマンスを実現する自動チューニングを実装するために **Performance Addon Operator** が使用されていました。**Red Hat OpenShift Service on AWS 4.11** 以降では、この機能は **Node Tuning Operator** の一部です。

6.2.1. Node Tuning Operator 仕様サンプルへのアクセス

このプロセスを使用して **Node Tuning Operator** 仕様サンプルにアクセスします。

手順

- 次のコマンドを実行して、NodeTuningOperator 仕様の例にアクセスします。

```
oc get tuned.tuned.openshift.io/default -o yaml -n openshift-cluster-node-tuning-operator
```

デフォルトの CR は、Red Hat OpenShift Service on AWS プラットフォームにノードレベルの標準のチューニングを提供することを目的としており、Operator 管理の状態を設定するためにのみ変更できます。デフォルト CR へのその他のカスタム変更は、Operator によって上書きされます。カスタムチューニングの場合は、独自のチューニングされた CR を作成します。新規に作成された CR は、ノード/Pod ラベルおよびプロファイルの優先順位に基づいて Red Hat OpenShift Service on AWS ノードに適用されるデフォルトの CR およびカスタムチューニングと組み合わせられます。



警告

特定の状況で Pod ラベルのサポートは必要なチューニングを自動的に配信する便利な方法ですが、この方法は推奨されず、とくに大規模なクラスターにおいて注意が必要です。デフォルトの調整された CR は Pod ラベル一致のない状態で提供されます。カスタムプロファイルが Pod ラベル一致のある状態で作成される場合、この機能はその時点で有効になります。Pod ラベル機能は、Node Tuning Operator の将来のバージョンで非推奨になる予定です。

6.2.2. カスタムチューニング仕様

Operator のカスタムリソース (CR) には 2 つの重要なセクションがあります。1 つ目のセクションの `profile:` は TuneD プロファイルおよびそれらの名前リストです。2 つ目の `recommend:` は、プロファイル選択ロジックを定義します。

複数のカスタムチューニング仕様は、Operator の namespace に複数の CR として共存できます。新規 CR の存在または古い CR の削除は Operator によって検出されます。既存のカスタムチューニング仕様はすべてマージされ、コンテナ化された TuneD デーモンの適切なオブジェクトは更新されません。

管理状態

Operator 管理の状態は、デフォルトの Tuned CR を調整して設定されます。デフォルトで、Operator は Managed 状態であり、`spec.managementState` フィールドはデフォルトの Tuned CR に表示されません。Operator Management 状態の有効な値は以下のとおりです。

- **Managed: Operator** は設定リソースが更新されるとそのオペランドを更新します。
- **Unmanaged: Operator** は設定リソースへの変更を無視します。
- **Removed: Operator** は **Operator** がプロビジョニングしたオペランドおよびリソースを削除します。

プロファイルデータ

profile: セクションは、TuneD プロファイルおよびそれらの名前をリスト表示します。

```
profile:
- name: tuned_profile_1
  data: |
    # TuneD profile specification
    [main]
    summary=Description of tuned_profile_1 profile

    [sysctl]
    net.ipv4.ip_forward=1
    # ... other sysctl's or other TuneD daemon plugins supported by the containerized TuneD

# ...

- name: tuned_profile_n
  data: |
    # TuneD profile specification
    [main]
    summary=Description of tuned_profile_n profile

    # tuned_profile_n profile settings
```

推奨プロファイル

profile: 選択ロジックは、CR の **recommend:** セクションによって定義されます。 **recommend:** セクションは、選択基準に基づくプロファイルの推奨項目のリストです。

```
recommend:
<recommend-item-1>
# ...
<recommend-item-n>
```

リストの個別項目:

```
- machineConfigLabels: ①
  <mcLabels> ②
  match: ③
  <match> ④
  priority: <priority> ⑤
  profile: <tuned_profile_name> ⑥
  operand: ⑦
  debug: <bool> ⑧
  tunedConfig:
    reapply_sysctl: <bool> ⑨
```

①

オプション:

②

キー/値の MachineConfig ラベルのディクショナリー。キーは一意である必要があります。

③

省略する場合は、優先度の高いプロファイルが最初に一致するか、machineConfigLabels が設定されていない限り、プロファイルの一致が想定されます。

④

オプションのリスト。

⑤

プロファイルの順序付けの優先度。数値が小さいほど優先度が高くなります (0 が最も高い優先度になります)。

⑥

一致に適用する TuneD プロファイル。例: tuned_profile_1

⑦

オプションのオペランド設定。

⑧

TuneD デーモンのデバッグオンまたはオフを有効にします。オプションは、オンの場合は true、オフの場合は false です。デフォルトは false です。

9

TuneD デーモンの `reapply_sysctl` 機能をオンまたはオフにします。オプションは on で true、オフの場合は false です。

<match> は、以下のように再帰的に定義されるオプションの一覧です。

```
- label: <label_name> ①
  value: <label_value> ②
  type: <label_type> ③
  <match> ④
```

1

ノードまたは Pod のラベル名。

2

オプションのノードまたは Pod のラベルの値。省略されている場合も、<label_name> があるだけで一致条件を満たします。

3

オプションのオブジェクトタイプ (node または pod)。省略されている場合は、node が想定されます。

4

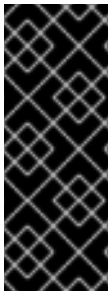
オプションの <match> リスト。

<match> が省略されない場合、ネストされたすべての <match> セクションが true に評価される必要があります。そうでない場合には false が想定され、それぞれの <match> セクションのあるプロファイルは適用されず、推奨されません。そのため、ネスト化 (子の <match> セクション) は論理 AND 演算子として機能します。これとは逆に、<match> 一覧のいずれかの項目が一致する場合は、<match> の一覧全体が true に評価されます。そのため、リストは論理 OR 演算子として機能します。

`machineConfigLabels` が定義されている場合は、マシン設定プールベースのマッチングが指定の `recommend`: 一覧の項目に対してオンになります。<mcLabels> はマシン設定のラベルを指定しま

す。マシン設定は、プロファイル `<tuned_profile_name>` についてカーネル起動パラメーターなどのホスト設定を適用するために自動的に作成されます。この場合は、マシン設定セレクターが `<mcLabels>` に一致するすべてのマシン設定プールを検索し、プロファイル `<tuned_profile_name>` を確認されるマシン設定プールが割り当てられるすべてのノードに設定する必要があります。マスターロールとワーカーのロールの両方を持つノードをターゲットにするには、マスターロールを使用する必要があります。

リスト項目の `match` および `machineConfigLabels` は論理 OR 演算子によって接続されます。 `match` 項目は、最初にショートサーキット方式で評価されます。そのため、`true` と評価される場合、`machineConfigLabels` 項目は考慮されません。



重要

マシン設定プールベースのマッチングを使用する場合は、同じハードウェア設定を持つノードを同じマシン設定プールにグループ化することが推奨されます。この方法に従わない場合は、TuneD オペランドが同じマシン設定プールを共有する 2 つ以上のノードの競合するカーネルパラメーターを計算する可能性があります。

例: ノードまたは Pod のラベルベースのマッチング

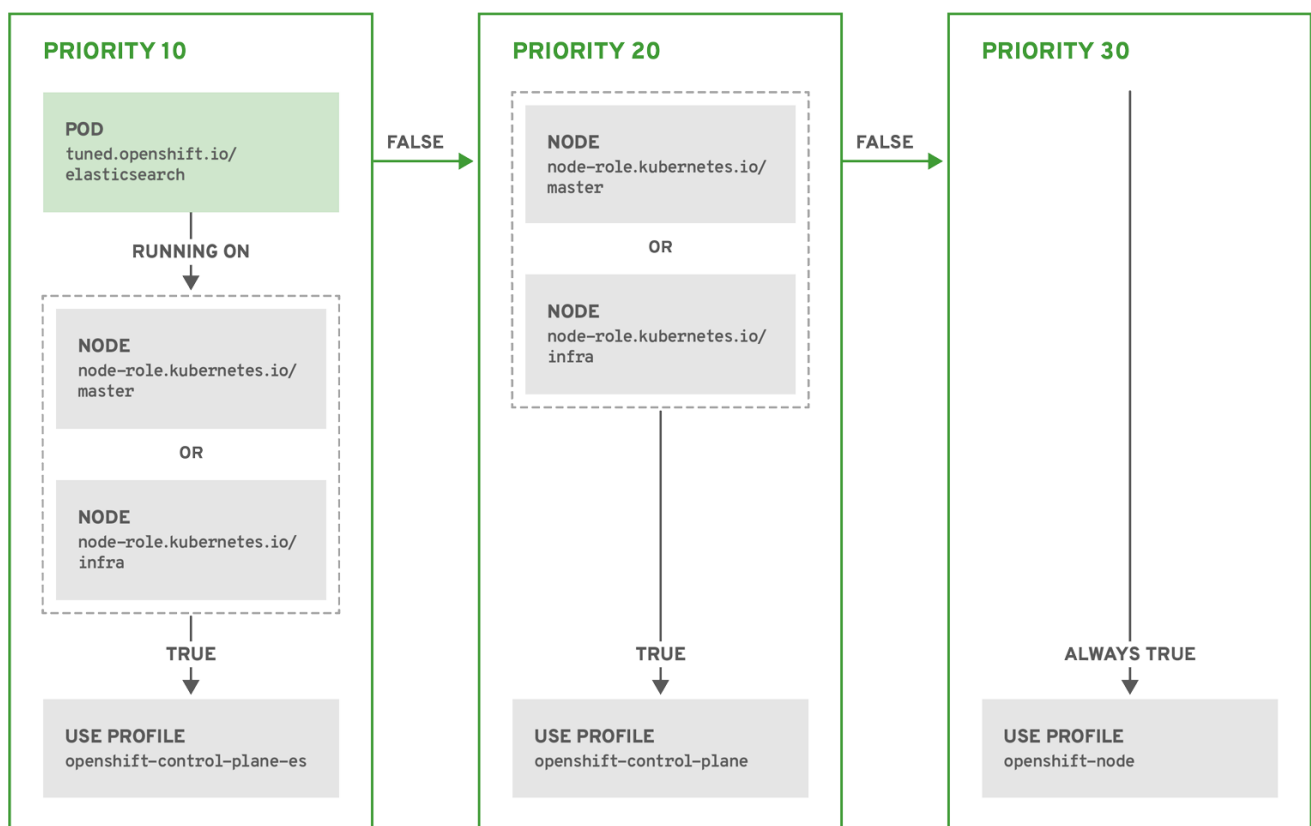
```
- match:
  - label: tuned.openshift.io/elasticsearch
    match:
      - label: node-role.kubernetes.io/master
      - label: node-role.kubernetes.io/infra
    type: pod
  priority: 10
  profile: openshift-control-plane-es
- match:
  - label: node-role.kubernetes.io/master
  - label: node-role.kubernetes.io/infra
  priority: 20
  profile: openshift-control-plane
- priority: 30
  profile: openshift-node
```

上記のコンテナ化された TuneD デーモンの CR は、プロファイルの優先順位に基づいてその `recommend.conf` ファイルに変換されます。最も高い優先順位 (10) を持つプロファイルは `openshift-control-plane-es` であるため、これが最初に考慮されます。指定されたノードで実行されるコンテナ化された TuneD デーモンは、同じノードに `tuned.openshift.io/elasticsearch` ラベルが設定された Pod が実行されているかどうかを確認します。これがない場合は、`<match>` セクション全体が `false` として評価されます。このラベルを持つこのような Pod がある場合に、`<match>` セクションが `true` に

評価されるようにするには、ノードラベルを `node-role.kubernetes.io/master` または `node-role.kubernetes.io/infra` にする必要もあります。

優先順位が 10 のプロファイルのラベルが一致した場合は、`openshift-control-plane-es` プロファイルが適用され、その他のプロファイルは考慮されません。ノード/Pod ラベルの組み合わせが一致しない場合は、2 番目に高い優先順位プロファイル (`openshift-control-plane`) が考慮されます。このプロファイルは、コンテナ化された TuneD Pod が `node-role.kubernetes.io/master` または `node-role.kubernetes.io/infra` ラベルを持つノードで実行される場合に適用されます。

最後に、プロファイル `openshift-node` には最低の優先順位である 30 が設定されます。これには `<match>` セクションがないため、常に一致します。これは、より高い優先順位の他のプロファイルが指定されたノードで一致しない場合に `openshift-node` プロファイルを設定するために、最低の優先順位のノードが適用される汎用的な (catch-all) プロファイルとして機能します。



OPENSIFT_10_0319

例: マシン設定プールベースのマッチング

```
apiVersion: tuned.openshift.io/v1
kind: Tuned
metadata:
  name: openshift-node-custom
  namespace: openshift-cluster-node-tuning-operator
```

```

spec:
  profile:
    - data: |
      [main]
      summary=Custom OpenShift node profile with an additional kernel parameter
      include=openshift-node
      [bootloader]
      cmdline_openshift_node_custom=+skew_tick=1
    name: openshift-node-custom

  recommend:
    - machineConfigLabels:
        machineconfiguration.openshift.io/role: "worker-custom"
      priority: 20
      profile: openshift-node-custom

```

ノードの再起動を最小限にするには、ターゲットノードにマシン設定プールのノードセレクターが一致するラベルを使用してラベルを付け、上記の Tuned CR を作成してから、最後にカスタムのマシン設定プール自体を作成します。

クラウドプロバイダー固有の TuneD プロファイル

この機能により、すべてのクラウドプロバイダー固有のノードに、Red Hat OpenShift Service on AWS クラスター上の特定のクラウドプロバイダーに合わせて特別に調整された TuneD プロファイルを簡単に割り当てることができます。これは、追加のノードラベルを追加したり、ノードをマシン設定プールにグループ化したりせずに実行できます。

この機能は、`<cloud-provider>://<cloud-provider-specific-id>` の形式で `spec.providerID` ノードオブジェクト値を利用して、NTO オペランドコンテナの `<cloud-provider>` の値で `/var/lib/tuned/provider` ファイルを書き込みます。その後、このファイルのコンテンツは TuneD により、プロバイダー `provider-<cloud-provider>` プロファイル (存在する場合) を読み込むために使用されます。

`openshift-control-plane` および `openshift-node` プロファイルの両方の設定を継承する `openshift` プロファイルは、条件付きプロファイルの読み込みを使用してこの機能を使用するよう更新されるようになりました。現時点で、NTO や TuneD にクラウドプロバイダー固有のプロファイルは含まれていません。ただし、すべてのクラウドプロバイダー固有のクラスターノードに適用されるカスタムプロファイル `provider-<cloud-provider>` を作成できます。

GCE クラウドプロバイダープロファイルの例


```

apiVersion: tuned.openshift.io/v1
kind: Tuned
metadata:
  name: provider-gce
  namespace: openshift-cluster-node-tuning-operator
spec:
  profile:
    - data: |
      [main]
      summary=GCE Cloud provider-specific profile
      # Your tuning for GCE Cloud provider goes here.
      name: provider-gce

```



注記

プロファイルの継承により、`provider-<cloud-provider>` プロファイルで指定された設定は、`openshift` プロファイルとその子プロファイルによって上書きされます。

6.2.3. クラスタに設定されるデフォルトのプロファイル

以下は、クラスタに設定されるデフォルトのプロファイルです。

```

apiVersion: tuned.openshift.io/v1
kind: Tuned
metadata:
  name: default
  namespace: openshift-cluster-node-tuning-operator
spec:
  profile:
    - data: |
      [main]
      summary=Optimize systems running OpenShift (provider specific parent profile)
      include--provider-{f:exec:cat:/var/lib/tuned/provider},openshift
      name: openshift
  recommend:
    - profile: openshift-control-plane
      priority: 30
    match:
      - label: node-role.kubernetes.io/master
      - label: node-role.kubernetes.io/infra
    - profile: openshift-node
      priority: 40

```

Red Hat OpenShift Service on AWS 4.9 以降では、すべての OpenShift TuneD プロファイルが TuneD パッケージに含まれています。oc exec コマンドを使用して、これらのプロファイルの内容を表

示できます。

```
$ oc exec $tuned_pod -n openshift-cluster-node-tuning-operator -- find /usr/lib/tuned/openshift{-control-plane,-node} -name tuned.conf -exec grep -H ^ {} \;
```

6.2.4. サポートされている TuneD デーモンプラグイン

[main] セクションを除き、以下の TuneD プラグインは、Tuned CR の profile: セクションで定義されたカスタムプロファイルを使用する場合にサポートされます。

- audio
- cpu
- disk
- eeepc_she
- modules
- mounts
- net
- scheduler
- scsi_host
- selinux
- sysctl

- **sysfs**
- **usb**
- **video**
- **vm**
- **bootloader**

これらのプラグインの一部によって提供される動的チューニング機能の中に、サポートされていない機能があります。以下の TuneD プラグインは現時点でサポートされていません。

- **script**
- **systemd**



注記

TuneD ブートローダープラグインは、Red Hat Enterprise Linux CoreOS (RHCOS) ワーカーノードのみサポートします。

関連情報

- [利用可能な TuneD プラグイン](#)
- [TuneD を使い始める](#)

6.3. ノードの修復、フェンシング、メンテナンス

カーネルのハングやネットワークインターフェイスコントローラー (NIC) の障害などをはじめとするノードレベルの障害が発生した場合、クラスターに必要な作業は減少せず、影響を受けたノードからの

ワークロードをどこかで再起動する必要があります。これらのワークロードに影響を与える障害は、データの損失、破損、またはその両方のリスクを伴います。ワークロードの復元 (remediation) とノードの復元を開始する前にノードを分離 (fencing) することが重要です。

ノードの修復、フェンシング、メンテナンスについて、詳しくは [Red Hat OpenShift のワークロードの可用性](#) を参照してください。

6.4. MACHINE CONFIG DAEMON メトリクス

Machine Config Daemon は Machine Config Operator の一部です。これはクラスター内のすべてのノードで実行されます。Machine Config Daemon は、各ノードの設定変更および更新を管理します。

6.4.1. Machine Config Daemon メトリクス

Red Hat OpenShift Service on AWS 4.3 以降、Machine Config Daemon はメトリクスのセットを提供します。これらのメトリクスには、Prometheus クラスターモニタリングスタックを使用してアクセスできます。

以下の表では、これらのメトリクスのセットについて説明しています。一部のエントリーには、特定のログを取得するためのコマンドが含まれています。ただし、最も包括的なログのセットは、`oc adm must-gather` コマンドを使用して入手できます。



注記

Name 列と Description 列に * が付いているメトリクスは、パフォーマンスの問題を引き起こす可能性のある重大なエラーを表します。このような問題により、更新およびアップグレードが続行されなくなる可能性があります。

表6.2 MCO メトリクス

| 名前 | フォーマット | 説明 | 備考 |
|--------------------------------------|--|---|----|
| <code>mcd_host_os_and_version</code> | <code>[]string{"os", "version"}</code> | RHCOS や RHEL など、MCD が実行されている OS を示します。RHCOS の場合、バージョンは指定されます。 | |

| 名前 | フォーマット | 説明 | 備考 |
|-----------------------------|--|--|--|
| <code>mcd_drain_err*</code> | | ドレイン (解放) の失敗時に受信されるエラーをログに記録します。* | ドレイン (解放) が成功するには、複数回試行する必要がある可能性があります。ターミナルでは、ドレイン (解放) に失敗すると更新を続行できなくなります。ドレイン (解放) にかかる時間を示す drain_time メトリクスはトラブルシューティングに役立つ可能性があります。 詳細な調査を実行するには、以下を実行してログを表示します。 \$ oc logs -f -n openshift-machine-config-operator machine-config-daemon- <hash> -c machine-config-daemon |
| <code>mcd_pivot_err*</code> | <code>[]string{"err", "node", "pivot_target"}</code> | ピボットで発生するログ。* | ピボットのエラーにより、OS のアップグレードを続行できなくなる可能性があります。 さらに調査するには、次のコマンドを実行して machine-config-daemon コンテナからのログを確認します。 \$ oc logs -f -n openshift-machine-config-operator machine-config-daemon- <hash> -c machine-config-daemon |
| <code>mcd_state</code> | <code>[]string{"state", "reason"}</code> | 指定ノードの Machine Config Daemon の状態。状態のオプションとして、Done、Working、および Degraded があります。Degraded の場合は、理由も含まれます。 | 詳細な調査を実行するには、以下を実行してログを表示します。 \$ oc logs -f -n openshift-machine-config-operator machine-config-daemon- <hash> -c machine-config-daemon |

| 名前 | フォーマット | 説明 | 備考 |
|---------------------------------|--|-----------------------------------|--|
| <code>mcd_kubelet_state*</code> | | kubelet の正常性についての失敗をログに記録します。* | <p>これは、失敗数が 0 で空になることが予想されます。失敗数が 2 を超えると、しきい値を超えたことを示すエラーが出されます。これは kubelet の正常性に関連した問題の可能性を示します。</p> <p>詳細な調査を行うには、以下のコマンドを実行してノードにアクセスし、そのすべてのログを表示します。</p> <pre>\$ oc debug node/<node> — chroot /host journalctl -u kubelet</pre> |
| <code>mcd_reboot_err*</code> | <code>[[string{"message", "err", "node"}]</code> | 再起動の失敗と対応するエラーをログに記録します。* | <p>これは空になることが予想されますが、これは再起動が成功したことを示します。</p> <p>詳細な調査を実行するには、以下を実行してログを表示します。</p> <pre>\$ oc logs -f -n openshift-machine-config-operator machine-config-daemon-<hash> -c machine-config-daemon</pre> |
| <code>mcd_update_state</code> | <code>[[string{"config", "err"}]</code> | 設定更新の成功または失敗、および対応するエラーをログに記録します。 | <p>予想される値は <code>rendered-master/rendered-worker-XXXX</code> です。更新に失敗すると、エラーが表示されます。</p> <p>詳細な調査を実行するには、以下を実行してログを表示します。</p> <pre>\$ oc logs -f -n openshift-machine-config-operator machine-config-daemon-<hash> -c machine-config-daemon</pre> |

関連情報

- [モニタリングスタックについて](#)

第7章 コンテナの使用

7.1. コンテナについて

Red Hat OpenShift Service on AWS アプリケーションの基本的な単位は コンテナ と呼ばれています。Linux コンテナテクノロジーは、指定されたリソースのみと対話するために実行中のプロセスを分離する軽量なメカニズムです。

数多くのアプリケーションインスタンスは、相互のプロセス、ファイル、ネットワークなどを可視化せずに単一ホストのコンテナで実行される可能性があります。通常、コンテナは任意のワークロードに使用されますが、各コンテナは Web サーバーまたはデータベースなどの (通常はマイクロサービスと呼ばれることの多い) 単一サービスを提供します。

Linux カーネルは数年にわたりコンテナテクノロジーの各種機能を統合してきました。Red Hat OpenShift Service on AWS および Kubernetes は、複数ホストのインストール間でコンテナのオーケストレーションを実行する機能を追加します。

7.1.1. コンテナおよび RHEL カーネルメモリーについて

Red Hat Enterprise Linux (RHEL) の動作により、CPU 使用率の高いノードのコンテナは、予想以上に多いメモリーを消費しているように見える可能性があります。メモリー消費量の増加は、RHEL カーネルの `kmem_cache` によって引き起こされる可能性があります。RHEL カーネルは、それぞれの `cgroup` に `kmem_cache` を作成します。パフォーマンスの強化のために、`kmem_cache` には `cpu_cache` と任意の NUMA ノードのノードキャッシュが含まれます。これらのキャッシュはすべてカーネルメモリーを消費します。

これらのキャッシュに保存されるメモリーの量は、システムが使用する CPU の数に比例します。結果として、CPU の数が増えると、より多くのカーネルメモリーがこれらのキャッシュに保持されます。これらのキャッシュ内のカーネルメモリーの量が増えると、Red Hat OpenShift Service on AWS コンテナが設定されたメモリー制限を超え、コンテナが強制終了される可能性があります。

カーネルメモリーの問題によりコンテナが失われないようにするには、コンテナが十分なメモリーを要求することを確認します。以下の式を使用して、`kmem_cache` が消費するメモリー量を見積ることができます。この場合、`nproc` は、`nproc` コマンドで報告される利用可能なプロセス数です。コンテナの要求の上限が低くなる場合、この値にコンテナメモリーの要件を加えた分になります。

```
$(nproc) X 1/2 MiB
```

7.1.2. コンテナエンジンとコンテナランタイムについて

コンテナエンジンは、コマンドラインオプションやイメージのプルなど、ユーザーの要求を処理するソフトウェアです。コンテナエンジンは、コンテナランタイム (下位レベルのコンテナランタイムとも呼ばれる) を使用して、コンテナのデプロイと操作に必要なコンポーネントを実行および管理します。コンテナエンジンまたはコンテナランタイムとやり取りする必要はほとんどありません。



注記

Red Hat OpenShift Service on AWS のドキュメントでは、コンテナランタイムという用語は、下位レベルのコンテナランタイムを指すために使用されています。他のドキュメントでは、コンテナエンジンをコンテナランタイムと呼んでいる場合があります。

Red Hat OpenShift Service on AWS は、コンテナエンジンとして CRI-O を使用し、コンテナランタイムとして runC または crun を使用します。デフォルトのコンテナランタイムは runC です。

7.2. POD のデプロイ前の、INIT コンテナの使用によるタスクの実行

Red Hat OpenShift Service on AWS は、init コンテナを提供します。このコンテナはアプリケーションコンテナの前に実行される特殊なコンテナであり、アプリケーションイメージに存在しないユーティリティーやセットアップスクリプトを含めることができます。

7.2.1. Init コンテナについて

Pod の残りの部分がデプロイされる前に、init コンテナリソースを使用して、タスクを実行することができます。

Pod は、アプリケーションコンテナに加えて、init コンテナを持つことができます。Init コンテナにより、セットアップスクリプトとバインディングコードを再編成できます。

init コンテナは以下のことを行うことができます。

- セキュリティ上の理由のためにアプリケーションコンテナイメージに含めることが望ましくないユーティリティーを含めることができ、それらを実行できます。
- アプリのイメージに存在しないセットアップに必要なユーティリティーまたはカスタムコードを含めることができます。たとえば、単に Sed、Awk、Python、Dig のようなツールを

セットアップ時に使用するために別のイメージからイメージを作成する必要はありません。

- **Linux namespace** を使用して、アプリケーションコンテナがアクセスできないシークレットへのアクセスなど、アプリケーションコンテナとは異なるファイルシステムビューを設定できます。

各 **init** コンテナは、次のコンテナが起動する前に正常に完了している必要があります。そのため、**init** コンテナには、一連の前提条件が満たされるまでアプリケーションコンテナの起動をブロックしたり、遅延させたりする簡単な方法となります。

たとえば、以下は **init** コンテナを使用するいくつかの方法になります。

- 以下のようなシェルコマンドでサービスが作成されるまで待機します。

```
for i in {1..100}; do sleep 1; if dig myservice; then exit 0; fi; done; exit 1
```

- 以下のようなコマンドを使用して、Downward API からリモートサーバーにこの Pod を登録します。

```
$ curl -X POST  
http://$MANAGEMENT_SERVICE_HOST:$MANAGEMENT_SERVICE_PORT/register -d  
'instance=${}&ip=${}'
```

- `sleep 60` のようなコマンドを使用して、アプリケーションコンテナが起動するまでしばらく待機します。
- Git リポジトリのクローンをボリュームに作成します。
- 設定ファイルに値を入力し、テンプレートツールを実行して、主要なアプリコンテナの設定ファイルを動的に生成します。たとえば、設定ファイルに `POD_IP` の値を入力し、**Jinja** を使用して主要なアプリ設定ファイルを生成します。

詳細は、[Kubernetes ドキュメント](#) を参照してください。

7.2.2. Init コンテナの作成

以下の例は、2つの init コンテナを持つ単純な Pod の概要を示しています。1つ目は `myservice` を待機し、2つ目は `mydb` を待機します。両方のコンテナが完了すると、Pod が開始されます。

手順

1. Init コンテナの Pod を作成します。
 - a. 以下のような YAML ファイルを作成します。

```
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod
  labels:
    app: myapp
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
  - name: myapp-container
    image: registry.access.redhat.com/ubi9/ubi:latest
    command: ['sh', '-c', 'echo The app is running! && sleep 3600']
    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
        drop: [ALL]
  - name: init-myservice
    image: registry.access.redhat.com/ubi9/ubi:latest
    command: ['sh', '-c', 'until getent hosts myservice; do echo waiting for
myservice; sleep 2; done;']
    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
        drop: [ALL]
  - name: init-mydb
    image: registry.access.redhat.com/ubi9/ubi:latest
    command: ['sh', '-c', 'until getent hosts mydb; do echo waiting for mydb; sleep
2; done;']
    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
        drop: [ALL]
```

- b. Pod を作成します。

```
$ oc create -f myapp.yaml
```

- c. Pod のステータスを表示します。

```
$ oc get pods
```

出力例

```
NAME           READY   STATUS    RESTARTS   AGE
myapp-pod      0/1     Init:0/2   0          5s
```

Pod のステータス `Init:0/2` は、2 つのサービスを待機していることを示します。

2. `myservice` サービスを作成します。

- a. 以下のような YAML ファイルを作成します。

```
kind: Service
apiVersion: v1
metadata:
  name: myservice
spec:
  ports:
  - protocol: TCP
    port: 80
    targetPort: 9376
```

- b. Pod を作成します。

```
$ oc create -f myservice.yaml
```

- c. Pod のステータスを表示します。

```
$ oc get pods
```

出力例

```

NAME                READY   STATUS    RESTARTS   AGE
myapp-pod           0/1    Init:1/2    0           5s

```

Pod のステータス `Init:1/2` は、1 つのサービス (この場合は `mydb` サービス) を待機していることを示します。

3.

`mydb` サービスを作成します。

a.

以下のような YAML ファイルを作成します。

```

kind: Service
apiVersion: v1
metadata:
  name: mydb
spec:
  ports:
  - protocol: TCP
    port: 80
    targetPort: 9377

```

b.

Pod を作成します。

```
$ oc create -f mydb.yaml
```

c.

Pod のステータスを表示します。

```
$ oc get pods
```

出力例

```

NAME                READY   STATUS    RESTARTS   AGE
myapp-pod           1/1    Running    0           2m

```

Pod のステータスは、サービスを待機しておらず、実行中であることを示していました。

7.3. ボリュームの使用によるコンテナデータの永続化

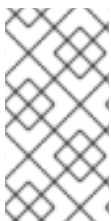
コンテナ内のファイルは一時的なものです。そのため、コンテナがクラッシュしたり停止したりした場合は、データが失われます。ボリュームを使用すると、Pod 内のコンテナが使用しているデータを永続化できます。ボリュームはディレクトリであり、Pod 内のコンテナからアクセスすることができます。ここでは、データが Pod の有効期間中保存されます。

7.3.1. ボリュームについて

ボリュームとは Pod およびコンテナで利用可能なマウントされたファイルシステムのことであり、これらは数多くのホストのローカルまたはネットワーク割り当てストレージのエンドポイントでサポートされる場合があります。コンテナはデフォルトで永続性がある訳ではなく、それらのコンテンツは再起動時にクリアされます。

ボリューム上のファイルシステムにエラーがないことを確認し、エラーが存在する場合は可能な限り修復するために、Red Hat OpenShift Service on AWS は `mount` ユーティリティより前に `fsck` ユーティリティを呼び出します。これはボリュームを追加するか、既存ボリュームを更新する際に実行されます。

最も単純なボリュームタイプは `emptyDir` です。これは、単一マシンの一時的なディレクトリです。管理者はユーザーによる Pod に自動的に割り当てられる永続ボリュームの要求を許可することもできます。



注記

`emptyDir` ボリュームストレージは、`FSGroup` パラメーターがクラスター管理者によって有効にされている場合は Pod の `FSGroup` に基づいてクォータで制限できます。

7.3.2. Red Hat OpenShift Service on AWS CLI を使用したボリュームの操作

CLI コマンド `oc set volume` を使用して、レプリケーションコントローラーやデプロイメント設定などの Pod テンプレートを持つオブジェクトのボリュームおよびボリュームマウントを追加し、削除

することができます。また、Pod または Pod テンプレートを持つオブジェクトのボリュームをリスト表示することもできます。

oc set volume コマンドは以下の一般的な構文を使用します。

```
$ oc set volume <object_selection> <operation> <mandatory_parameters> <options>
```

オブジェクトの選択

oc set volume コマンドの object_selection パラメーターに、以下のいずれかを指定します。

表7.1 オブジェクトの選択

| 構文 | 説明 | 例 |
|---|---|--|
| <object_type> <name> | タイプ <object_type> の <name> を選択します。 | deploymentConfig registry |
| <object_type>/<name> | タイプ <object_type> の <name> を選択します。 | deploymentConfig/registry |
| <object_type>--selector=<object_label_selector> | 所定のラベルセクターに一致するタイプ <object_type> のリソースを選択します。 | deploymentConfig--selector="name=registry" |
| <object_type> --all | タイプ <object_type> のすべてのリソースを選択します。 | deploymentConfig --all |
| -f または --filename=<file_name> | リソースを編集するために使用するファイル名、ディレクトリー、または URL です。 | -f registry-deployment-config.json |

操作

oc set volume コマンドの operation パラメーターに --add または --remove を指定します。

必須パラメーター

いずれの必須パラメーターも選択された操作に固有のものであり、これらについては後のセクションで説明します。

オプション

いずれのオプションも選択された操作に固有のものであり、これらについては後のセクションで説明します。

7.3.3. Pod のボリュームとボリュームマウントのリスト表示

Pod または Pod テンプレートのボリュームおよびボリュームマウントをリスト表示することができます。

手順

ボリュームをリスト表示するには、以下の手順を実行します。

```
$ oc set volume <object_type>/<name> [options]
```

ボリュームのサポートされているオプションをリスト表示します。

| オプション | 説明 | デフォルト |
|-------------------------|--|-------|
| --name | ボリュームの名前。 | |
| -c, --containers | 名前でコンテナを選択します。すべての文字に一致するワイルドカード '*' を取ることもできます。 | '*' |

以下に例を示します。

- Pod p1 のすべてのボリュームをリスト表示するには、以下を実行します。

```
$ oc set volume pod/p1
```

- すべてのデプロイメント設定で定義されるボリューム v1 をリスト表示するには、以下の手順を実行します。

```
$ oc set volume dc --all --name=v1
```

7.3.4. Pod へのボリュームの追加

Pod にボリュームとボリュームマウントを追加することができます。

手順

ボリューム、ボリュームマウントまたはそれらの両方を Pod テンプレートに追加するには、以下を実行します。

```
$ oc set volume <object_type>/<name> --add [options]
```

表7.2 ボリュームを追加するためのサポートされるオプション

| オプション | 説明 | デフォルト |
|-------------------------|--|----------------------|
| --name | ボリュームの名前。 | 指定がない場合は、自動的に生成されます。 |
| -t, --type | ボリュームソースの名前。サポートされる値は emptyDir 、 hostPath 、 secret 、 configmap 、 persistentVolumeClaim または projected です。 | emptyDir |
| -c, --containers | 名前でコンテナを選択します。すべての文字に一致するワイルドカード ** を取ることもできます。 | ** |
| -m, --mount-path | 選択されたコンテナ内のマウントパス。コンテナのルート (/) や、ホストとコンテナで同じパスにはマウントしないでください。これは、コンテナに十分な特権が付与されている場合、ホストシステムを破壊する可能性があります (例: ホストの /dev/pts ファイル)。ホストをマウントするには、 /host を使用するのが安全です。 | |
| --path | ホストパス。 --type=hostPath の必須パラメーターです。コンテナのルート (/) や、ホストとコンテナで同じパスにはマウントしないでください。これは、コンテナに十分な特権が付与されている場合、ホストシステムを破壊する可能性があります (例: ホストの /dev/pts ファイル)。ホストをマウントするには、 /host を使用するのが安全です。 | |
| --secret-name | シークレットの名前。 --type=secret の必須パラメーターです。 | |

| オプション | 説明 | デフォルト |
|-------------------------|---|--------------------|
| --configmap-name | configmap の名前。 -- type=configmap の必須のパラ メーターです。 | |
| --claim-name | 永続ボリューム要求 (PVC) の名 前。 -- type=persistentVolumeClaim の必須パラメーターです。 | |
| --source | JSON 文字列としてのボリューム ソースの詳細。必要なボリューム ソースが -- type でサポートされ ない場合に推奨されます。 | |
| -o, --output | サーバー上で更新せずに変更した オブジェクトを表示します。サ ポートされる値は json 、 yaml で す。 | |
| --output-version | 指定されたバージョンで変更され たオブジェクトを出力します。 | api-version |

以下に例を示します。

- 新規ボリュームソース `emptyDir` を `registry DeploymentConfig` オブジェクトに追加するには、以下を実行します。

```
$ oc set volume dc/registry --add
```

ヒント

あるいは、以下の YAML を適用してボリュームを追加できます。

例7.1 ボリュームを追加したデプロイメント設定の例

```
kind: DeploymentConfig
apiVersion: apps.openshift.io/v1
metadata:
  name: registry
  namespace: registry
spec:
  replicas: 3
  selector:
    app: httpd
  template:
    metadata:
      labels:
        app: httpd
    spec:
      volumes: ①
      - name: volume-pppsw
        emptyDir: {}
    containers:
      - name: httpd
        image: >-
          image-registry.openshift-image-registry.svc:5000/openshift/httpd:latest
        ports:
          - containerPort: 8080
            protocol: TCP
```

①

ボリュームソース `emptyDir` を追加します。

•

レプリケーションコントローラー `r1` のシークレット `secret1` を使用してボリューム `v1` を追加し、コンテナ内の `/data` でマウントするには、以下を実行します。

```
$ oc set volume rc/r1 --add --name=v1 --type=secret --secret-name='secret1' --mount-path=/data
```

ヒント

あるいは、以下の YAML を適用してボリュームを追加できます。

例7.2 ボリュームおよびシークレットを追加したレプリケーションコントローラーの例

```
kind: ReplicationController
apiVersion: v1
metadata:
  name: example-1
  namespace: example
spec:
  replicas: 0
  selector:
    app: httpd
    deployment: example-1
    deploymentconfig: example
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: httpd
        deployment: example-1
        deploymentconfig: example
    spec:
      volumes: ①
      - name: v1
        secret:
          secretName: secret1
          defaultMode: 420
      containers:
      - name: httpd
        image: >-
          image-registry.openshift-image-registry.svc:5000/openshift/httpd:latest
        volumeMounts: ②
      - name: v1
        mountPath: /data
```

①

ボリュームおよびシークレットを追加します。

②

コンテナのマウントパスを追加します。

- 要求名 pvc1 を使用して既存の永続ボリューム v1 をディスク上のデプロイメント設定 dc.json に追加し、ボリュームをコンテナ c1 の /data にマウントし、サーバー上で DeploymentConfig オブジェクトを更新します。

```
$ oc set volume -f dc.json --add --name=v1 --type=persistentVolumeClaim \
--claim-name=pvc1 --mount-path=/data --containers=c1
```

ヒント

あるいは、以下の YAML を適用してボリュームを追加できます。

例7.3 永続ボリュームが追加されたデプロイメント設定の例

```
kind: DeploymentConfig
apiVersion: apps.openshift.io/v1
metadata:
  name: example
  namespace: example
spec:
  replicas: 3
  selector:
    app: httpd
  template:
    metadata:
      labels:
        app: httpd
    spec:
      volumes:
        - name: volume-pppsw
          emptyDir: {}
        - name: v1 1
          persistentVolumeClaim:
            claimName: pvc1
      containers:
        - name: httpd
          image: >-
            image-registry.openshift-image-registry.svc:5000/openshift/httpd:latest
          ports:
            - containerPort: 8080
              protocol: TCP
          volumeMounts: 2
            - name: v1
              mountPath: /data
```

1

'pvc1' という名前の永続ボリューム要求を追加します。

2

コンテナのマウントパスを追加します。

すべてのレプリケーションコントローラー向けにリビジョン 5125c45f9f563 を使い、Git リポジトリ <https://github.com/namespace1/project1> に基づいてボリューム v1 を追加するには、以下の手順を実行します。

```
$ oc set volume rc --all --add --name=v1 \  
  --source='{ "gitRepo": {  
    "repository": "https://github.com/namespace1/project1",  
    "revision": "5125c45f9f563"  
  } }'
```

7.3.5. Pod 内のボリュームとボリュームマウントの更新

Pod 内のボリュームとボリュームマウントを変更することができます。

手順

--overwrite オプションを使用して、既存のボリュームを更新します。

```
$ oc set volume <object_type>/<name> --add --overwrite [options]
```

以下に例を示します。

- レプリケーションコントローラー r1 の既存ボリューム v1 を既存の永続ボリューム要求 (PVC) pvc1 に置き換えるには、以下の手順を実行します。

```
$ oc set volume rc/r1 --add --overwrite --name=v1 --type=persistentVolumeClaim --  
claim-name=pvc1
```

ヒント

または、以下の YAML を適用してボリュームを置き換えることもできます。

例7.4 pvc1という名前の永続ボリューム要求を持つレプリケーションコントローラーの例

```
kind: ReplicationController
apiVersion: v1
metadata:
  name: example-1
  namespace: example
spec:
  replicas: 0
  selector:
    app: httpd
    deployment: example-1
    deploymentconfig: example
  template:
    metadata:
      labels:
        app: httpd
        deployment: example-1
        deploymentconfig: example
    spec:
      volumes:
        - name: v1 1
          persistentVolumeClaim:
            claimName: pvc1
      containers:
        - name: httpd
          image: >-
            image-registry.openshift-image-registry.svc:5000/openshift/httpd:latest
          ports:
            - containerPort: 8080
              protocol: TCP
          volumeMounts:
            - name: v1
              mountPath: /data
```

1

永続ボリューム要求を pvc1 に設定します。

•

DeploymentConfig オブジェクトの d1 のマウントポイントを、ボリューム v1 の /opt に変更するには、以下を実行します。

```
$ oc set volume dc/d1 --add --overwrite --name=v1 --mount-path=/opt
```

ヒント

または、以下の YAML を適用してマウントポイントを変更できます。

例7.5 マウントポイントがoptに設定されたデプロイメント設定の例

```
kind: DeploymentConfig
apiVersion: apps.openshift.io/v1
metadata:
  name: example
  namespace: example
spec:
  replicas: 3
  selector:
    app: httpd
  template:
    metadata:
      labels:
        app: httpd
    spec:
      volumes:
        - name: volume-pppsw
          emptyDir: {}
        - name: v2
          persistentVolumeClaim:
            claimName: pvc1
        - name: v1
          persistentVolumeClaim:
            claimName: pvc1
      containers:
        - name: httpd
          image: >-
            image-registry.openshift-image-registry.svc:5000/openshift/httpd:latest
          ports:
            - containerPort: 8080
              protocol: TCP
          volumeMounts: ①
            - name: v1
              mountPath: /opt
```

①

マウントポイントを /opt に設定します。

7.3.6. Pod からのボリュームおよびボリュームマウントの削除

Pod からボリュームまたはボリュームマウントを削除することができます。

手順

Pod テンプレートからボリュームを削除するには、以下を実行します。

```
$ oc set volume <object_type>/<name> --remove [options]
```

表7.3 ボリュームを削除するためにサポートされるオプション

| オプション | 説明 | デフォルト |
|-------------------------|---|--------------------|
| --name | ボリュームの名前。 | |
| -c, --containers | 名前でコンテナを選択します。すべての文字に一致するワイルドカード '*' を取ることもできます。 | '*' |
| --confirm | 複数のボリュームを1度に削除することを示します。 | |
| -o, --output | サーバー上で更新せずに変更したオブジェクトを表示します。サポートされる値は json 、 yaml です。 | |
| --output-version | 指定されたバージョンで変更されたオブジェクトを出力します。 | api-version |

以下に例を示します。

- DeploymentConfig オブジェクトの d1 から ボリューム v1 を削除するには、以下を実行します。

```
$ oc set volume dc/d1 --remove --name=v1
```

- DeploymentConfig オブジェクトの d1 の c1 のコンテナからボリューム v1 をアンマウントし、d1 のコンテナで参照されていない場合にボリューム v1 を削除するには、以下の手順を実行します。

```
$ oc set volume dc/d1 --remove --name=v1 --containers=c1
```

- レプリケーションコントローラー r1 のすべてのボリュームを削除するには、以下の手順を実行します。


```
$ oc set volume rc/r1 --remove --confirm
```

7.3.7. Pod 内での複数の用途のためのボリュームの設定

ボリュームを、単一 Pod で複数の使用目的のためにボリュームを共有するように設定できます。この場合、`volumeMounts.subPath` プロパティを使用し、ボリュームのルートの代わりにボリューム内に `subPath` 値を指定します。



注記

既存のスケジュールされた Pod に `subPath` パラメーターを追加することはできません。

手順

1. ボリューム内のファイルのリストを表示するには、`oc rsh` コマンドを実行します。

```
$ oc rsh <pod>
```

出力例

```
sh-4.2$ ls /path/to/volume/subpath/mount
example_file1 example_file2 example_file3
```

2. `subPath` を指定します。

`subPath` パラメーターを含む Pod 仕様の例

```
apiVersion: v1
kind: Pod
metadata:
  name: my-site
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
```

```
type: RuntimeDefault
containers:
- name: mysql
  image: mysql
  volumeMounts:
  - mountPath: /var/lib/mysql
    name: site-data
    subPath: mysql ①
  securityContext:
    allowPrivilegeEscalation: false
  capabilities:
    drop: [ALL]
- name: php
  image: php
  volumeMounts:
  - mountPath: /var/www/html
    name: site-data
    subPath: html ②
  securityContext:
    allowPrivilegeEscalation: false
  capabilities:
    drop: [ALL]
volumes:
- name: site-data
  persistentVolumeClaim:
    claimName: my-site-data
```

①

データベースは `mysql` フォルダに保存されます。

②

HTML コンテンツは `html` フォルダに保存されます。

7.4. PROJECTED ポリリュームによるポリリュームのマッピング

Projected ポリリュームは、いくつかの既存のポリリュームソースを同じディレクトリーにマップします。

以下のタイプのポリリュームソースをデプロイメントできます。

- シークレット
- Config Map
- Downward API



注記

すべてのソースは Pod と同じ namespace に置かれる必要があります。

7.4.1. Projected ポリ्यूームについて

Projected ポリ्यूームはこれらのポリ्यूームソースの任意の組み合わせを単一ディレクトリーにマップし、ユーザーの以下の実行を可能にします。

- 単一ポリ्यूームを、複数のシークレットのキー、設定マップ、および Downward API 情報で自動的に設定し、各種の情報ソースで単一ディレクトリーを合成できるようにします。
- 各項目のパスを明示的に指定して、単一ポリ्यूームを複数シークレットのキー、設定マップ、および Downward API 情報で設定し、ユーザーがポリ्यूームの内容を完全に制御できるようにします。



重要

RunAsUser パーミッションが Linux ベースの Pod のセキュリティーコンテキストに設定されている場合、Projected ファイルには、コンテナユーザー所有権を含む適切なパーミッションが設定されます。ただし、Windows の同等の RunAsUsername パーミッションが Windows Pod に設定されている場合、kubelet は Projected ポリ्यूームのファイルに正しい所有権を設定できません。

そのため、Windows Pod のセキュリティーコンテキストに設定された RunAsUsername パーミッションは、Red Hat OpenShift Service on AWS で実行される Windows の Projected ポリ्यूームには適用されません。

以下の一般的なシナリオは、Projected ポリ्यूームを使用する方法について示しています。

設定マップ、シークレット、Downward API

Projected ボリュームを使用すると、パスワードが含まれる設定データでコンテナをデプロイできます。これらのリソースを使用するアプリケーションは、Red Hat OpenStack Platform (RHOSP) を Kubernetes にデプロイしている可能性があります。設定データは、サービスが稼働用またはテストで使用されるかによって異なった方法でアセンブルされる必要がある可能性があります。Pod に稼働用またはテストのラベルが付けられている場合、Downward API セレクター `metadata.labels` を使用して適切な RHOSP 設定を生成できます。

設定マップ + シークレット

Projected ボリュームにより、設定データおよびパスワードを使用してコンテナをデプロイできます。たとえば、設定マップを、Vault パスワードファイルを使用して暗号解除する暗号化された機密タスクで実行する場合があります。

ConfigMap + Downward API.

Projected ボリュームにより、Pod 名 (`metadata.name` セレクターで選択可能) を含む設定を生成できます。このアプリケーションは IP トラッキングを使用せずに簡単にソースを判別できるよう要求と共に Pod 名を渡すことができます。

シークレット + Downward API

Projected ボリュームにより、Pod の namespace (`metadata.namespace` セレクターで選択可能) を暗号化するためのパブリックキーとしてシークレットを使用できます。この例では、Operator はこのアプリケーションを使用し、暗号化されたトランスポートを使用せずに namespace 情報を安全に送信できるようになります。

7.4.1.1. Pod 仕様の例

以下は、Projected ボリュームを作成するための Pod 仕様の例です。

シークレット、Downward API および設定マップを含む Pod

```
apiVersion: v1
kind: Pod
metadata:
  name: volume-test
spec:
  securityContext:
    runAsNonRoot: true
  seccompProfile:
    type: RuntimeDefault
  containers:
  - name: container-test
    image: busybox
    volumeMounts: ①
```

```
- name: all-in-one
  mountPath: "/projected-volume" 2
  readOnly: true 3
securityContext:
  allowPrivilegeEscalation: false
capabilities:
  drop: [ALL]
volumes: 4
- name: all-in-one 5
  projected:
    defaultMode: 0400 6
    sources:
      - secret:
          name: mysecret 7
          items:
            - key: username
              path: my-group/my-username 8
      - downwardAPI: 9
          items:
            - path: "labels"
              fieldRef:
                fieldPath: metadata.labels
            - path: "cpu_limit"
              resourceFieldRef:
                containerName: container-test
                resource: limits.cpu
      - configMap: 10
          name: myconfigmap
          items:
            - key: config
              path: my-group/my-config
              mode: 0777 11
```

1

シークレットを必要とする各コンテナの `volumeMounts` セクションを追加します。

2

シークレットが表示される未使用ディレクトリーのパスを指定します。

3

`readOnly` を `true` に設定します。

4

5

ボリュームの名前を指定します。

6

ファイルに実行パーミッションを設定します。

7

シークレットを追加します。シークレットオブジェクトの名前を追加します。使用する必要のあるそれぞれのシークレットはリスト表示される必要があります。

8

`mountPath` の下にシークレットへのパスを指定します。ここで、シークレットファイルは `/projected-volume/my-group/my-username` になります。

9

Downward API ソースを追加します。

10

ConfigMap ソースを追加します。

11

特定のデプロイメントにおけるモードを設定します。



注記

Pod に複数のコンテナがある場合、それぞれのコンテナには `volumeMounts` セクションが必要ですが、1つの `volumes` セクションのみが必要になります。

デフォルト以外のパーミッションモデルが設定された複数シークレットを含む Pod

```
apiVersion: v1
kind: Pod
metadata:
  name: volume-test
```

```

spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
  - name: container-test
    image: busybox
    volumeMounts:
    - name: all-in-one
      mountPath: "/projected-volume"
      readOnly: true
    securityContext:
      allowPrivilegeEscalation: false
    capabilities:
      drop: [ALL]
  volumes:
  - name: all-in-one
    projected:
      defaultMode: 0755
      sources:
      - secret:
          name: mysecret
          items:
            - key: username
              path: my-group/my-username
      - secret:
          name: mysecret2
          items:
            - key: password
              path: my-group/my-password
              mode: 511

```



注記

`defaultMode` はデプロイメントされるレベルでのみ指定でき、各ボリュームソースには指定されません。ただし、上記のように個々のデプロイメントについての `mode` を明示的に指定できます。

7.4.1.2. パスについての留意事項

設定されるパスが同一である場合のキー間の競合

複数のキーを同じパスで設定する場合、Pod 仕様は有効な仕様として受け入れられません。以下の例では、`mysecret` および `myconfigmap` に指定されるパスは同じです。

```

apiVersion: v1
kind: Pod

```

```
metadata:
  name: volume-test
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
  - name: container-test
    image: busybox
    volumeMounts:
    - name: all-in-one
      mountPath: "/projected-volume"
      readOnly: true
    securityContext:
      allowPrivilegeEscalation: false
    capabilities:
      drop: [ALL]
  volumes:
  - name: all-in-one
    projected:
      sources:
      - secret:
          name: mysecret
          items:
          - key: username
            path: my-group/data
      - configMap:
          name: myconfigmap
          items:
          - key: config
            path: my-group/data
```

ボリュームファイルのパスに関連する以下の状況を検討しましょう。

設定されたパスのないキー間の競合

上記のシナリオの場合と同様に、実行時の検証が実行される唯一のタイミングはすべてのパスが Pod の作成時に認識される時です。それ以外の場合は、競合の発生時に指定された最新のリソースがこれより前のすべてのものを上書きします (これは Pod 作成後に更新されるリソースについても同様です)。

1つのパスが明示的なパスであり、もう1つのパスが自動的にデプロイメントされるパスである場合の競合

自動的にデプロイメントされるデータに一致するユーザー指定パスによって競合が生じる場合、前述のように後からのリソースがこれより前のすべてのものを上書きします。

7.4.2. Pod の Projected ボリュームの設定

Projected ボリュームを作成する場合は、Pprojected ボリュームについてで説明されているボリュームファイルパスの状態を考慮します。

以下の例では、Projected ボリュームを使用して、既存のシークレットボリュームソースをマウントする方法が示されています。以下の手順は、ローカルファイルからユーザー名およびパスワードのシークレットを作成するために実行できます。その後、シークレットを同じ共有ディレクトリーにマウントするために Projected ボリュームを使用して1つのコンテナを実行する Pod を作成します。

このユーザー名とパスワードの値には、base64 でエンコードされた任意の有効な文字列を使用できます。

以下の例は、base64 の admin を示しています。

```
$ echo -n "admin" | base64
```

出力例

```
YWRtaW4=
```

以下の例は、base64 のパスワード 1f2d1e2e67df を示しています。

```
$ echo -n "1f2d1e2e67df" | base64
```

出力例

```
MWYyZDFIMmU2N2Rm
```

手順

既存のシークレットボリュームソースをマウントするために Projected ボリュームを使用するには、以下を実行します。

1. シークレットを作成します。
 - a. 次のような YAML ファイルを作成し、パスワードとユーザー情報を適切に置き換えます。

```
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
type: Opaque
data:
  pass: MWYyZDFIMmU2N2Rm
  user: YWRtaW4=
```

- b. 以下のコマンドを使用してシークレットを作成します。

```
$ oc create -f <secrets-filename>
```

以下に例を示します。

```
$ oc create -f secret.yaml
```

出力例

```
secret "mysecret" created
```

- c. シークレットが以下のコマンドを使用して作成されていることを確認できます。

```
$ oc get secret <secret-name>
```

以下に例を示します。

```
$ oc get secret mysecret
```

出力例

```
NAME      TYPE      DATA      AGE
mysecret  Opaque    2          17h
```

```
$ oc get secret <secret-name> -o yaml
```

以下に例を示します。

```
$ oc get secret mysecret -o yaml
```

```
apiVersion: v1
data:
  pass: MWYyZDFIMmU2N2Rm
  user: YWRtaW4=
kind: Secret
metadata:
  creationTimestamp: 2017-05-30T20:21:38Z
  name: mysecret
  namespace: default
  resourceVersion: "2107"
  selfLink: /api/v1/namespaces/default/secrets/mysecret
  uid: 959e0424-4575-11e7-9f97-fa163e4bd54c
type: Opaque
```

2. 投影されたボリュームを持つ Pod を作成します。

- a. `volumes` セクションを含む、次のような YAML ファイルを作成します。

```
kind: Pod
metadata:
  name: test-projected-volume
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
  - name: test-projected-volume
    image: busybox
    args:
    - sleep
```

```
- "86400"  
volumeMounts:  
- name: all-in-one  
  mountPath: "/projected-volume"  
  readOnly: true  
securityContext:  
  allowPrivilegeEscalation: false  
  capabilities:  
    drop: [ALL]  
volumes:  
- name: all-in-one  
  projected:  
    sources:  
    - secret:  
      name: mysecret ①
```

①

作成されたシークレットの名前。

b.

設定ファイルから Pod を作成します。

```
$ oc create -f <your_yaml_file>.yaml
```

以下に例を示します。

```
$ oc create -f secret-pod.yaml
```

出力例

```
pod "test-projected-volume" created
```

3.

Pod コンテナが実行中であることを確認してから、Pod への変更を確認します。

```
$ oc get pod <name>
```

以下に例を示します。

```
$ oc get pod test-projected-volume
```

出力は以下のようになります。

出力例

```
NAME                READY   STATUS    RESTARTS   AGE
test-projected-volume 1/1     Running   0           14s
```

4.

別のターミナルで、`oc exec` コマンドを使用し、実行中のコンテナに対してシェルを開きます。

```
$ oc exec -it <pod> <command>
```

以下に例を示します。

```
$ oc exec -it test-projected-volume -- /bin/sh
```

5.

シェルで、`projected-volumes` ディレクトリーにデプロイメントされるソースが含まれることを確認します。

```
/ # ls
```

出力例

```
bin      home      root      tmp
dev      proc      run       usr
etc      projected-volume sys       var
```

7.5. コンテナによる API オブジェクト使用の許可

Downward API は、コンテナが Red Hat OpenShift Service on AWS に接続せずに API オブジェクトに関する情報を利用できるようにするメカニズムです。この情報には、Pod の名前、namespace およびリソース値が含まれます。コンテナは、環境変数やボリュームプラグインを使用して Downward API からの情報を使用できます。

7.5.1. Downward API の使用によるコンテナへの Pod 情報の公開

Downward API には、Pod の名前、プロジェクト、リソースの値などの情報が含まれます。コンテナは、環境変数やボリュームプラグインを使用して Downward API からの情報を使用できます。

Pod 内のフィールドは、FieldRef API タイプを使用して選択されます。FieldRef には 2 つのフィールドがあります。

| フィールド | 説明 |
|-------------------|---|
| fieldPath | Pod に関連して選択するフィールドのパスです。 |
| apiVersion | fieldPath セレクターの解釈に使用する API バージョンです。 |

現時点で v1 API の有効なセレクターには以下が含まれます。

| セレクター | 説明 |
|-----------------------------|---|
| metadata.name | Pod の名前です。これは環境変数およびボリュームでサポートされています。 |
| metadata.namespace | Pod の namespace です。これは環境変数およびボリュームでサポートされています。 |
| metadata.labels | Pod のラベルです。これはボリュームでのみサポートされ、環境変数ではサポートされていません。 |
| metadata.annotations | Pod のアノテーションです。これはボリュームでのみサポートされ、環境変数ではサポートされていません。 |
| status.podIP | Pod の IP です。これは環境変数でのみサポートされ、ボリュームではサポートされていません。 |

apiVersion フィールドは、指定されていない場合は、対象の Pod テンプレートの API バージョン

にデフォルト設定されます。

7.5.2. Downward API を使用してコンテナの値を使用する方法について

コンテナは、環境変数やボリュームプラグインを使用して API の値を使用することができます。選択する方法により、コンテナは以下を使用できます。

- Pod の名前
- Pod プロジェクト/namespace
- Pod のアノテーション
- Pod のラベル

アノテーションとラベルは、ボリュームプラグインのみを使用して利用できます。

7.5.2.1. 環境変数の使用によるコンテナ値の使用

コンテナの環境変数を設定する際に、EnvVar タイプの `valueFrom` フィールド (タイプは `EnvVarSource`) を使用して、変数の値が `value` フィールドで指定されるリテラル値ではなく、`FieldRef` ソースからの値になるように指定します。

この方法で使用できるのは Pod の定数属性のみです。変数の値の変更についてプロセスに通知する方法でプロセスを起動すると、環境変数を更新できなくなるためです。環境変数を使用してサポートされるフィールドには、以下が含まれます。

- Pod の名前
- Pod プロジェクト/namespace

手順

1. コンテナで使用する環境変数を含む新しい Pod 仕様を作成します。
 - a. 次のような pod.yaml ファイルを作成します。

```
apiVersion: v1
kind: Pod
metadata:
  name: dapi-env-test-pod
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
  - name: env-test-container
    image: gcr.io/google_containers/busybox
    command: [ "/bin/sh", "-c", "env" ]
    env:
    - name: MY_POD_NAME
      valueFrom:
        fieldRef:
          fieldPath: metadata.name
    - name: MY_POD_NAMESPACE
      valueFrom:
        fieldRef:
          fieldPath: metadata.namespace
    securityContext:
      allowPrivilegeEscalation: false
    capabilities:
      drop: [ALL]
  restartPolicy: Never
# ...
```

- b. pod.yaml ファイルから Pod を作成します。

```
$ oc create -f pod.yaml
```

検証

- コンテナのログで MY_POD_NAME および MY_POD_NAMESPACE の値を確認します。

```
$ oc logs -p dapi-env-test-pod
```

7.5.2.2. ボリュームプラグインを使用したコンテナ値の使用

コンテナは、ボリュームプラグインを使用して API 値を使用できます。

コンテナは、以下を使用できます。

- Pod の名前
- Pod プロジェクト/namespace
- Pod のアノテーション
- Pod のラベル

手順

ボリュームプラグインを使用するには、以下の手順を実行します。

1. コンテナで使用する環境変数を含む新しい Pod 仕様を作成します。
 - a. 次のような volume-pod.yaml ファイルを作成します。

```
kind: Pod
apiVersion: v1
metadata:
  labels:
    zone: us-east-coast
    cluster: downward-api-test-cluster1
    rack: rack-123
  name: dapi-volume-test-pod
  annotations:
    annotation1: "345"
    annotation2: "456"
spec:
  securityContext:
    runAsNonRoot: true
  seccompProfile:
    type: RuntimeDefault
  containers:
  - name: volume-test-container
    image: gcr.io/google_containers/busybox
```

```

command: ["sh", "-c", "cat /tmp/etc/pod_labels /tmp/etc/pod_annotations"]
volumeMounts:
  - name: podinfo
    mountPath: /tmp/etc
    readOnly: false
securityContext:
  allowPrivilegeEscalation: false
  capabilities:
    drop: [ALL]
volumes:
  - name: podinfo
    downwardAPI:
      defaultMode: 420
      items:
        - fieldRef:
            fieldPath: metadata.name
            path: pod_name
        - fieldRef:
            fieldPath: metadata.namespace
            path: pod_namespace
        - fieldRef:
            fieldPath: metadata.labels
            path: pod_labels
        - fieldRef:
            fieldPath: metadata.annotations
            path: pod_annotations
      restartPolicy: Never
# ...

```

b.

volume-pod.yaml ファイルから Pod を作成します。

```
$ oc create -f volume-pod.yaml
```

検証

- コンテナのログを確認し、設定されたフィールドの有無を確認します。

```
$ oc logs -p dapi-volume-test-pod
```

出力例

```

cluster=downward-api-test-cluster1
rack=rack-123
zone=us-east-coast
annotation1=345
annotation2=456
kubernetes.io/config.source=api

```

7.5.3. Downward API を使用してコンテナリソースを使用する方法について

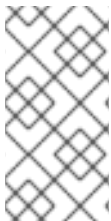
Pod の作成時に、Downward API を使用してコンピューティングリソースの要求および制限についての情報を挿入し、イメージおよびアプリケーションの作成者が特定の環境用のイメージを適切に作成できるようにします。

環境変数またはボリュームプラグインを使用してこれを実行できます。

7.5.3.1. 環境変数を使用したコンテナリソースの使用

Pod を作成するときは、Downward API を使用し、環境変数を使用してコンピューティングリソースの要求と制限に関する情報を挿入できます。

Pod 設定の作成時に、spec.container フィールド内の resources フィールドの内容に対応する環境変数を指定します。



注記

リソース制限がコンテナ設定に含まれていない場合、Downward API はデフォルトでノードの CPU およびメモリーの割り当て可能な値に設定されます。

手順

1. 注入するリソースを含む新しい Pod 仕様を作成します。
 - a. 次のような pod.yaml ファイルを作成します。

```
apiVersion: v1
kind: Pod
metadata:
  name: dapi-env-test-pod
spec:
  containers:
  - name: test-container
    image: gcr.io/google_containers/busybox:1.24
    command: [ "/bin/sh", "-c", "env" ]
    resources:
```

```

requests:
  memory: "32Mi"
  cpu: "125m"
limits:
  memory: "64Mi"
  cpu: "250m"
env:
- name: MY_CPU_REQUEST
  valueFrom:
    resourceFieldRef:
      resource: requests.cpu
- name: MY_CPU_LIMIT
  valueFrom:
    resourceFieldRef:
      resource: limits.cpu
- name: MY_MEM_REQUEST
  valueFrom:
    resourceFieldRef:
      resource: requests.memory
- name: MY_MEM_LIMIT
  valueFrom:
    resourceFieldRef:
      resource: limits.memory
# ...

```

b.

pod.yaml ファイルから Pod を作成します。

```
$ oc create -f pod.yaml
```

7.5.3.2. ボリュームプラグインを使用したコンテナリソースの使用

Pod を作成するときは、Downward API を使用し、ボリュームプラグインを使用してコンピューティングリソースの要求と制限に関する情報を挿入できます。

Pod 設定の作成時に、spec.volumes.downwardAPI.items フィールドを使用して spec.resources フィールドに対応する必要なリソースを記述します。



注記

リソース制限がコンテナ設定に含まれていない場合、Downward API はデフォルトでノードの CPU およびメモリーの割り当て可能な値に設定されます。

手順

1.

注入するリソースを含む新しい Pod 仕様を作成します。

a.

次のような pod.yaml ファイルを作成します。

```
apiVersion: v1
kind: Pod
metadata:
  name: dapi-env-test-pod
spec:
  containers:
    - name: client-container
      image: gcr.io/google_containers/busybox:1.24
      command: ["sh", "-c", "while true; do echo; if [[ -e /etc/cpu_limit ]]; then cat /etc/cpu_limit; fi; if [[ -e /etc/cpu_request ]]; then cat /etc/cpu_request; fi; if [[ -e /etc/mem_limit ]]; then cat /etc/mem_limit; fi; if [[ -e /etc/mem_request ]]; then cat /etc/mem_request; fi; sleep 5; done"]
      resources:
        requests:
          memory: "32Mi"
          cpu: "125m"
        limits:
          memory: "64Mi"
          cpu: "250m"
      volumeMounts:
        - name: podinfo
          mountPath: /etc
          readOnly: false
  volumes:
    - name: podinfo
      downwardAPI:
        items:
          - path: "cpu_limit"
            resourceFieldRef:
              containerName: client-container
              resource: limits.cpu
          - path: "cpu_request"
            resourceFieldRef:
              containerName: client-container
              resource: requests.cpu
          - path: "mem_limit"
            resourceFieldRef:
              containerName: client-container
              resource: limits.memory
          - path: "mem_request"
            resourceFieldRef:
              containerName: client-container
              resource: requests.memory
# ...
```

b.

volume-pod.yaml ファイルから Pod を作成します。

```
$ oc create -f volume-pod.yaml
```

7.5.4. Downward API を使用したシークレットの使用

Pod の作成時に、Downward API を使用してシークレットを挿入し、イメージおよびアプリケーションの作成者が特定の環境用のイメージを作成できるようにできます。

手順

1. 注入するシークレットを作成します。
 - a. 次のような Secret.yaml ファイルを作成します。

```
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
data:
  password: <password>
  username: <username>
type: kubernetes.io/basic-auth
```

- b. Secret.yaml ファイルからシークレットオブジェクトを作成します。

```
$ oc create -f secret.yaml
```

2. 上記の Secret オブジェクトから username フィールドを参照する Pod を作成します。
 - a. 次のような pod.yaml ファイルを作成します。

```
apiVersion: v1
kind: Pod
metadata:
  name: dapi-env-test-pod
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
    - name: env-test-container
      image: gcr.io/google_containers/busybox
```

```
command: [ "/bin/sh", "-c", "env" ]
env:
  - name: MY_SECRET_USERNAME
    valueFrom:
      secretKeyRef:
        name: mysecret
        key: username
  securityContext:
    allowPrivilegeEscalation: false
  capabilities:
    drop: [ALL]
  restartPolicy: Never
# ...
```

- b. pod.yaml ファイルから Pod を作成します。

```
$ oc create -f pod.yaml
```

検証

- コンテナのログで MY_SECRET_USERNAME の値を確認します。

```
$ oc logs -p dapi-env-test-pod
```

7.5.5. Downward API を使用した設定マップの使用

Pod の作成時に、Downward API を使用して設定マップの値を挿入し、イメージおよびアプリケーションの作成者が特定の環境用のイメージを作成することができるようにすることができます。

手順

1. 注入する値を含む config map を作成します。
 - a. 次のような configmap.yaml ファイルを作成します。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: myconfigmap
data:
  mykey: myvalue
```

- b. configmap.yaml ファイルから config map を作成します。

```
$ oc create -f configmap.yaml
```

2. 上記の config map を参照する Pod を作成します。

- a. 次のような pod.yaml ファイルを作成します。

```
apiVersion: v1
kind: Pod
metadata:
  name: dapi-env-test-pod
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
    - name: env-test-container
      image: gcr.io/google_containers/busybox
      command: [ "/bin/sh", "-c", "env" ]
      env:
        - name: MY_CONFIGMAP_VALUE
          valueFrom:
            configMapKeyRef:
              name: myconfigmap
              key: mykey
      securityContext:
        allowPrivilegeEscalation: false
      capabilities:
        drop: [ALL]
      restartPolicy: Always
# ...
```

- b. pod.yaml ファイルから Pod を作成します。

```
$ oc create -f pod.yaml
```

検証

- コンテナのログで MY_CONFIGMAP_VALUE の値を確認します。

```
$ oc logs -p dapi-env-test-pod
```


7.5.6. 環境変数の参照

Pod の作成時に、`$()` 構文を使用して事前に定義された環境変数の値を参照できます。環境変数の参照が解決されない場合、値は提供された文字列のままになります。

手順

1. 既存の環境変数を参照する Pod を作成します。
 - a. 次のような `pod.yaml` ファイルを作成します。

```
apiVersion: v1
kind: Pod
metadata:
  name: dapi-env-test-pod
spec:
  securityContext:
    runAsNonRoot: true
  seccompProfile:
    type: RuntimeDefault
  containers:
  - name: env-test-container
    image: gcr.io/google_containers/busybox
    command: [ "/bin/sh", "-c", "env" ]
    env:
    - name: MY_EXISTING_ENV
      value: my_value
    - name: MY_ENV_VAR_REF_ENV
      value: $(MY_EXISTING_ENV)
    securityContext:
      allowPrivilegeEscalation: false
    capabilities:
      drop: [ALL]
  restartPolicy: Never
# ...
```

- b. `pod.yaml` ファイルから Pod を作成します。

```
$ oc create -f pod.yaml
```

検証

- コンテナのログで `MY_ENV_VAR_REF_ENV` 値を確認します。

```
$ oc logs -p dapi-env-test-pod
```

-

7.5.7. 環境変数の参照のエスケープ

Pod の作成時に、二重ドル記号を使用して環境変数の参照をエスケープできます。次に値は指定された値の単一ドル記号のバージョンに設定されます。

手順

1. 既存の環境変数を参照する Pod を作成します。
 - a. 次のような pod.yaml ファイルを作成します。

```
apiVersion: v1
kind: Pod
metadata:
  name: dapi-env-test-pod
spec:
  securityContext:
    runAsNonRoot: true
  seccompProfile:
    type: RuntimeDefault
  containers:
  - name: env-test-container
    image: gcr.io/google_containers/busybox
    command: [ "/bin/sh", "-c", "env" ]
    env:
    - name: MY_NEW_ENV
      value: $$SOME_OTHER_ENV
    securityContext:
      allowPrivilegeEscalation: false
    capabilities:
      drop: [ALL]
  restartPolicy: Never
# ...
```

- b. pod.yaml ファイルから Pod を作成します。

```
$ oc create -f pod.yaml
```

検証

- コンテナのログで MY_NEW_ENV 値を確認します。

```
$ oc logs -p dapi-env-test-pod
```

7.6. RED HAT OPENSIFT SERVICE ON AWS コンテナ間でのファイルのコピー

CLI を使用して、`rsync` コマンドでコンテナのリモートディレクトリーにローカルファイルをコピーするか、そのディレクトリーからローカルファイルをコピーすることができます。

7.6.1. ファイルをコピーする方法について

`oc rsync` コマンドまたは `remote sync` は、バックアップと復元を実行するためにデータベースアーカイブを Pod にコピー、または Pod からコピーするのに役立つツールです。また、実行中の Pod がソースファイルのホットリロードをサポートする場合に、ソースコードの変更を開発のデバッグ目的で実行中の Pod にコピーするためにも、`oc rsync` を使用できます。

```
$ oc rsync <source> <destination> [-c <container>]
```

7.6.1.1. 要件

Copy Source の指定

`oc rsync` コマンドのソース引数はローカルディレクトリーまたは Pod ディレクトリーのいずれかを示す必要があります。個々のファイルはサポートされていません。

Pod ディレクトリーを指定する場合、ディレクトリー名の前に Pod 名を付ける必要があります。

```
<pod name>:<dir>
```

ディレクトリー名がパスセパレーター (`/`) で終了する場合、ディレクトリーの内容のみが宛先にコピーされます。それ以外の場合は、ディレクトリーとその内容が宛先にコピーされます。

Copy Destination の指定

`oc rsync` コマンドの宛先引数はディレクトリーを参照する必要があります。ディレクトリーが存在せず、`rsync` がコピーに使用される場合、ディレクトリーが作成されます。

宛先でのファイルの削除

`--delete` フラグは、ローカルディレクトリーにないリモートディレクトリーにあるファイルを削除するために使用できます。

ファイル変更についての継続的な同期

`--watch` オプションを使用すると、コマンドはソースパスでファイルシステムの変更をモニターし、変更が生じるとそれらを同期します。この引数を指定すると、コマンドは無期限に実行さ

れます。

同期は短い非表示期間の後に実行され、急速に変化するファイルシステムによって同期呼び出しが継続的に実行されないようにします。

--watch オプションを使用する場合、動作は通常 `oc rsync` に渡される引数の使用を含め `oc rsync` を繰り返し手動で起動する場合と同様になります。そのため、--delete などの `oc rsync` の手動の呼び出しで使用される同じフラグでこの動作を制御できます。

7.6.2. コンテナへの/からのファイルのコピー

コンテナへの/からのローカルファイルのコピーのサポートは CLI に組み込まれています。

前提条件

`oc rsync` を使用する場合は、以下の点に注意してください。

- `rsync` がインストールされていること。`oc rsync` コマンドは、クライアントマシンおよびリモートコンテナ上に存在する場合は、ローカルの `rsync` ツールを使用。

`rsync` がローカルの場所またはリモートコンテナに見つからない場合は、`tar` アーカイブがローカルに作成されてからコンテナに送信されます。ここで、`tar` ユーティリティーがファイルのデプロイメントに使用されます。リモートコンテナで `tar` を利用できない場合は、コピーに失敗します。

`tar` のコピー方法は `oc rsync` と同様に機能する訳ではありません。たとえば、`oc rsync` は、宛先ディレクトリが存在しない場合にはこれを作成し、ソースと宛先間の差分のファイルのみを送信します。



注記

Windows では、`cwRsync` クライアントが `oc rsync` コマンドで使用するためにインストールされ、`PATH` に追加される必要があります。

手順

- ローカルディレクトリを Pod ディレクトリにコピーするには、以下の手順を実行します。

```
$ oc rsync <local-dir> <pod-name>:/<remote-dir> -c <container-name>
```

以下に例を示します。

```
$ oc rsync /home/user/source devpod1234:/src -c user-container
```

- Pod ディレクトリーをローカルディレクトリーにコピーするには、以下の手順を実行します。

```
$ oc rsync devpod1234:/src /home/user/source
```

出力例

```
$ oc rsync devpod1234:/src/status.txt /home/user/
```

7.6.3. 高度な Rsync 機能の使用

oc rsync コマンドは標準の rsync よりも少ないコマンドラインのオプションを表示します。oc rsync で利用できない標準の rsync コマンドラインオプションを使用する必要がある場合 (例: --exclude-from=FILE オプション)、以下のように回避策として標準 rsync の --rsh (-e) オプション、または RSYNC_RSH 環境変数を使用できる場合があります。

```
$ rsync --rsh='oc rsh' --exclude-from=<file_name> <local-dir> <pod-name>:/<remote-dir>
```

または、以下を実行します。

RSYNC_RSH 変数をエクスポートします。

```
$ export RSYNC_RSH='oc rsh'
```

次に、rsync コマンドを実行します。

```
$ rsync --exclude-from=<file_name> <local-dir> <pod-name>:/<remote-dir>
```

上記の例のいずれも標準の `rsync` をリモートシェルプログラムとして `oc rsh` を使用するように設定してリモート Pod に接続できるようにします。これらは `oc rsync` を実行する代替方法となります。

7.7. RED HAT OPENSIFT SERVICE ON AWS コンテナでのリモートコマンドの実行

CLI を使用して、Red Hat OpenShift Service on AWS コンテナでリモートコマンドを実行できます。

7.7.1. コンテナでのリモートコマンドの実行

リモートコンテナコマンドの実行についてサポートは CLI に組み込まれています。

手順

コンテナでコマンドを実行するには、以下の手順を実行します。

```
$ oc exec <pod> [-c <container>] -- <command> [<arg_1> ... <arg_n>]
```

以下に例を示します。

```
$ oc exec mypod date
```

出力例

```
Thu Apr 9 02:21:53 UTC 2015
```



重要

セキュリティ保護の理由 により、`oc exec` コマンドは、コマンドが `cluster-admin` ユーザーによって実行されている場合を除き、特権付きコンテナにアクセスしようとしても機能しません。

7.7.2. クライアントからのリモートコマンドを開始するためのプロトコル

クライアントは要求を Kubernetes API サーバーに対して実行してコンテナのリモートコマンドの実行を開始します。

```
/proxy/nodes/<node_name>/exec/<namespace>/<pod>/<container>?command=<command>
```

上記の URL には以下が含まれます。

- `<node_name>` はノードの FQDN です。
- `<namespace>` はターゲット Pod のプロジェクトです。
- `<pod>` はターゲット Pod の名前です。
- `<container>` はターゲットコンテナの名前です。
- `<command>` は実行される必要なコマンドです。

以下に例を示します。

```
/proxy/nodes/node123.openshift.com/exec/myns/mypod/mycontainer?command=date
```

さらに、クライアントはパラメーターを要求に追加して以下について指示します。

- クライアントはリモートクライアントのコマンドに入力を送信する (標準入力: `stdin`)。
- クライアントのターミナルは TTY である。
- リモートコンテナのコマンドは標準出力 (`stdout`) からクライアントに出力を送信する。
-

リモートコンテナのコマンドは標準エラー出力 (stderr) からクライアントに出力を送信する。

exec 要求の API サーバーへの送信後、クライアントは多重化ストリームをサポートするものに接続をアップグレードします。現在の実装では HTTP/2 を使用しています。

クライアントは標準入力 (stdin)、標準出力 (stdout)、および標準エラー出力 (stderr) 用にそれぞれのストリームを作成します。ストリームを区別するために、クライアントはストリームの streamType ヘッダーを stdin、stdout、または stderr のいずれかに設定します。

リモートコマンド実行要求の処理が終了すると、クライアントはすべてのストリームやアップグレードされた接続および基礎となる接続を閉じます。

7.8. コンテナ内のアプリケーションにアクセスするためのポート転送の使用

Red Hat OpenShift Service on AWS は Pod へのポート転送をサポートしています。

7.8.1. ポート転送について

CLI を使用して 1 つ以上のローカルポートを Pod に転送できます。これにより、指定されたポートまたはランダムポートでローカルにリッスンでき、Pod の所定ポートへからデータを転送できます。

ポート転送のサポートは、CLI に組み込まれています。

```
$ oc port-forward <pod> [<local_port>:]<remote_port> [...<local_port_n>:]<remote_port_n>]
```

CLI はユーザーによって指定されたそれぞれのローカルポートでリッスンし、以下で説明されているプロトコルで転送を実行します。

ポートは以下の形式を使用して指定できます。

| | |
|------------------|--|
| 5000 | クライアントはポート 5000 でローカルにリッスンし、Pod の 5000 に転送します。 |
| 6000:5000 | クライアントはポート 6000 でローカルにリッスンし、Pod の 5000 に転送します。 |

| | |
|---------------------|--|
| :5000 または 0:5000 | クライアントは空きのローカルポートを選択し、Pod の 5000 に転送します。 |
|---------------------|--|

Red Hat OpenShift Service on AWS はクライアントからのポート転送要求を処理します。要求を受信すると、Red Hat OpenShift Service on AWS は応答をアップグレードし、クライアントがポート転送ストリームを作成するまで待機します。Red Hat OpenShift Service on AWS は新しいストリームを受信すると、ストリームと Pod のポートの間でデータをコピーします。

アーキテクチャーの観点では、Pod のポートに転送するためのいくつかのオプションがあります。サポートされている Red Hat OpenShift Service on AWS 実装は、ノードホストで `nsenter` を直接呼び出して Pod のネットワーク namespace に入ってから、`socat` を呼び出してストリームと Pod のポートの間でデータをコピーします。ただし、カスタムの実装には、`nsenter` および `socat` を実行する helper Pod の実行を含めることができ、その場合は、それらのバイナリーをホストにインストールする必要はありません。

7.8.2. ポート転送の使用

CLI を使用して、1 つ以上のローカルポートの Pod へのポート転送を実行できます。

手順

以下のコマンドを使用して、Pod 内の指定されたポートでリッスンします。

```
$ oc port-forward <pod> [<local_port>:]<remote_port> [...[<local_port_n>:]<remote_port_n>]
```

以下に例を示します。

- 以下のコマンドを使用して、ポート 5000 および 6000 でローカルにリッスンし、Pod のポート 5000 および 6000 との間でデータを転送します。

```
$ oc port-forward <pod> 5000 6000
```

出力例

```
Forwarding from 127.0.0.1:5000 -> 5000
Forwarding from [::1]:5000 -> 5000
Forwarding from 127.0.0.1:6000 -> 6000
```

Forwarding from [::1]:6000 -> 6000

- 以下のコマンドを使用して、ポート 8888 でローカルにリッスンし、Pod の 5000 に転送します。

```
$ oc port-forward <pod> 8888:5000
```

出力例

```
Forwarding from 127.0.0.1:8888 -> 5000  
Forwarding from [::1]:8888 -> 5000
```

- 以下のコマンドを使用して、空きポートでローカルにリッスンし、Pod の 5000 に転送します。

```
$ oc port-forward <pod> :5000
```

出力例

```
Forwarding from 127.0.0.1:42390 -> 5000  
Forwarding from [::1]:42390 -> 5000
```

または、以下を実行します。

```
$ oc port-forward <pod> 0:5000
```

7.8.3. クライアントからのポート転送を開始するためのプロトコル

クライアントは Kubernetes API サーバーに対して要求を実行して Pod へのポート転送を実行しま

す。

```
/proxy/nodes/<node_name>/portForward/<namespace>/<pod>
```

上記の URL には以下が含まれます。

- `<node_name>` はノードの FQDN です。
- `<namespace>` はターゲット Pod の namespace です。
- `<pod>` はターゲット Pod の名前です。

以下に例を示します。

```
/proxy/nodes/node123.openshift.com/portForward/myns/mypod
```

ポート転送要求を API サーバーに送信した後に、クライアントは多重化ストリームをサポートするものに接続をアップグレードします。現在の実装では [Hypertext Transfer Protocol Version 2 \(HTTP/2\)](#) を使用しています。

クライアントは Pod のターゲットポートを含む `port` ヘッダーでストリームを作成します。ストリームに書き込まれるすべてのデータは kubelet 経由でターゲット Pod およびポートに送信されます。同様に、転送された接続で Pod から送信されるすべてのデータはクライアントの同じストリームに送信されます。

クライアントは、ポート転送要求が終了するとすべてのストリーム、アップグレードされた接続および基礎となる接続を閉じます。

第8章 クラスターの操作

8.1. RED HAT OPENSIFT SERVICE ON AWS クラスター内のシステムイベント情報の表示

Red Hat OpenShift Service on AWS のイベントは、Red Hat OpenShift Service on AWS クラスター内の API オブジェクトに発生するイベントに基づいてモデル化されます。

8.1.1. イベントについて

イベントにより、Red Hat OpenShift Service on AWS はリソースに依存しない方法で実際のイベントについての情報を記録できます。また、開発者および管理者が統一された方法でシステムコンポーネントについての情報を使用できるようにします。

8.1.2. CLI を使用したイベントの表示

CLI を使用し、特定のプロジェクト内のイベントのリストを取得できます。

手順

- プロジェクト内のイベントを表示するには、以下のコマンドを使用します。

```
$ oc get events [-n <project>] ①
```

①

プロジェクトの名前。

以下に例を示します。

```
$ oc get events -n openshift-config
```

出力例

```
LAST SEEN   TYPE      REASON          OBJECT                    MESSAGE
97m         Normal    Scheduled       pod/dapi-env-test-pod    Successfully
assigned openshift-config/dapi-env-test-pod to ip-10-0-171-202.ec2.internal
97m         Normal    Pulling         pod/dapi-env-test-pod    pulling image
"gcr.io/google_containers/busybox"
```

```

97m      Normal  Pulled           pod/dapi-env-test-pod    Successfully pulled
image "gcr.io/google_containers/busybox"
97m      Normal  Created          pod/dapi-env-test-pod    Created container
9m5s     Warning  FailedCreatePodSandBox pod/dapi-volume-test-pod Failed
create pod sandbox: rpc error: code = Unknown desc = failed to create pod network
sandbox k8s_dapi-volume-test-pod_openshift-config_6bc60c1f-452e-11e9-9140-
0eec59c23068_0(748c7a40db3d08c07fb4f9eba774bd5effe5f0d5090a242432a73eee66ba9
e22): Multus: Err adding pod to network "openshift-sdn": cannot set "openshift-sdn"
ifname to "eth0": no netns: failed to Statfs "/proc/33366/ns/net": no such file or
directory
8m31s    Normal  Scheduled        pod/dapi-volume-test-pod Successfully
assigned openshift-config/dapi-volume-test-pod to ip-10-0-171-202.ec2.internal

```

-

Red Hat OpenShift Service on AWS コンソールからプロジェクト内のイベントを表示するには、以下を実行します。

1. Red Hat OpenShift Service on AWS コンソールを起動します。
2. Home → Events をクリックし、プロジェクトを選択します。
3. イベントを表示するリソースに移動します。たとえば、Home → Projects → <project-name> → <resource-name> の順に移動します。

Pod や デプロイメントなどの多くのオブジェクトには、独自の イベント タブもあります。それらのタブには、オブジェクトに関連するイベントが表示されます。

8.1.3. イベントのリスト

このセクションでは、Red Hat OpenShift Service on AWS のイベントについて説明します。

表8.1 設定イベント

| 名前 | 説明 |
|------------------|-------------------|
| FailedValidation | Pod 設定の検証に失敗しました。 |

表8.2 コンテナイベント

| 名前 | 説明 |
|-----------------------------|---------------------------------------|
| BackOff | バックオフ (再起動) によりコンテナが失敗しました。 |
| Created | コンテナが作成されました。 |
| Failed | プル/作成/起動が失敗しました。 |
| Killing | コンテナを強制終了しています。 |
| Started | コンテナが起動しました。 |
| Preempting | 他の Pod のプリエンプションを実行します。 |
| ExceededGrace Period | コンテナランタイムは、指定の猶予期間以内に Pod を停止しませんでした。 |

表8.3 正常性に関するイベント

| 名前 | 説明 |
|------------------|-----------------|
| Unhealthy | コンテナが正常ではありません。 |

表8.4 イメージイベント

| 名前 | 説明 |
|--------------------------|--------------------------------------|
| BackOff | バックオフ (コンテナ起動、イメージのプル)。 |
| ErrImageNeverPull | イメージの NeverPull Policy の違反があります。 |
| Failed | イメージのプルに失敗しました。 |
| InspectFailed | イメージの検査に失敗しました。 |
| Pulled | イメージのプルに成功し、コンテナイメージがマシンにすでに置かれています。 |
| Pulling | イメージをプルしています。 |

表8.5 イメージマネージャーイベント

| 名前 | 説明 |
|----------------------------|-------------------------|
| FreeDiskSpaceFailed | 空きディスク容量に関連する障害が発生しました。 |

| 名前 | 説明 |
|----------------------------|--------------|
| InvalidDiskCapacity | 無効なディスク容量です。 |

表8.6 ノードイベント

| 名前 | 説明 |
|--------------------------------|-------------------------|
| FailedMount | ボリュームのマウントに失敗しました。 |
| HostNetworkNotSupported | ホストのネットワークがサポートされていません。 |
| HostPortConflict | ホスト/ポートの競合 |
| KubeletSetupFailed | Kubelet のセットアップに失敗しました。 |
| NilShaper | シェイパーが定義されていません。 |
| NodeNotReady | ノードの準備ができていません。 |
| NodeNotSchedulable | ノードがスケジュール可能ではありません。 |
| NodeReady | ノードの準備ができています。 |
| NodeSchedulable | ノードがスケジュール可能です。 |
| NodeSelectorMismatching | ノードセレクターの不一致があります。 |
| OutOfDisk | ディスクの空き容量が不足しています。 |
| Rebooted | ノードが再起動しました。 |
| Starting | kubelet を起動しています。 |
| FailedAttachVolume | ボリュームの割り当てに失敗しました。 |
| FailedDetachVolume | ボリュームの割り当て解除に失敗しました。 |

| 名前 | 説明 |
|---|------------------------------|
| VolumeResizeFailed | ボリュームの拡張/縮小に失敗しました。 |
| VolumeResizeSuccessful | 正常にボリュームを拡張/縮小しました。 |
| FileSystemResizeFailed | ファイルシステムの拡張/縮小に失敗しました。 |
| FileSystemResizeSuccessful | 正常にファイルシステムが拡張/縮小されました。 |
| FailedUnmount | ボリュームのマウント解除に失敗しました。 |
| FailedMapVolume | ボリュームのマッピングに失敗しました。 |
| FailedUnmapDevice | デバイスのマッピング解除に失敗しました。 |
| AlreadyMountedVolume | ボリュームがすでにマウントされています。 |
| SuccessfulDetachVolume | ボリュームの割り当てが正常に解除されました。 |
| SuccessfulMountVolume | ボリュームが正常にマウントされました。 |
| SuccessfulUnmountVolume | ボリュームのマウントが正常に解除されました。 |
| ContainerGCFailed | コンテナのガベージコレクションに失敗しました。 |
| ImageGCFailed | イメージのガベージコレクションに失敗しました。 |
| FailedNodeAllocatableEnforcement | システム予約の Cgroup 制限の実施に失敗しました。 |
| NodeAllocatableEnforced | システム予約の Cgroup 制限を有効にしました。 |

| 名前 | 説明 |
|-------------------------------|--------------------------|
| UnsupportedMountOption | マウントオプションが非対応です。 |
| SandboxChanged | Pod のサンドボックスが変更されました。 |
| FailedCreatePodSandbox | Pod のサンドボックスの作成に失敗しました。 |
| FailedPodSandboxStatus | Pod サンドボックスの状態取得に失敗しました。 |

表8.7 Pod ワーカーイベント

| 名前 | 説明 |
|-------------------|-----------------|
| FailedSync | Pod の同期が失敗しました。 |

表8.8 システムイベント

| 名前 | 説明 |
|------------------|---------------------------------------|
| SystemOOM | クラスターに OOM (out of memory) 状態が発生しました。 |

表8.9 Pod に関するイベント

| 名前 | 説明 |
|---------------------------------|------------------------------------|
| FailedKillPod | Pod の停止に失敗しました。 |
| FailedCreatePodContainer | Pod コンテナの作成に失敗しました。 |
| Failed | Pod データディレクトリーの作成に失敗しました。 |
| NetworkNotReady | ネットワークの準備ができていません。 |
| FailedCreate | 作成エラー: <error-msg> |
| SuccessfulCreate | 作成された Pod: <pod-name> |
| FailedDelete | 削除エラー: <error-msg> |

| 名前 | 説明 |
|-------------------------|---------------------------------|
| SuccessfulDelete | 削除した Pod: <pod-id> |

表8.10 Horizontal Pod Autoscaler に関するイベント

| 名前 | 説明 |
|-------------------------------------|--|
| SelectorRequired | セレクターが必要です。 |
| InvalidSelector | セレクターを適切な内部セレクターオブジェクトに変換できませんでした。 |
| FailedGetObjectMetric | HPA はレプリカ数を計算できませんでした。 |
| InvalidMetricSourceType | 不明なメトリクスソースタイプです。 |
| ValidMetricFound | HPA は正常にレプリカ数を計算できました。 |
| FailedConvertHPA | 指定の HPA への変換に失敗しました。 |
| FailedGetScale | HPA コントローラーは、ターゲットの現在のスケールを取得できませんでした。 |
| SucceededGetScale | HPA コントローラーは、ターゲットの現在のスケールを取得できました。 |
| FailedComputeMetricsReplicas | 表示されているメトリクスに基づく必要なレプリカ数の計算に失敗しました。 |
| FailedRescale | 新しいサイズ: <size> ; 理由: <msg> ; エラー: <error-msg> |
| SuccessfulRescale | 新しいサイズ: <size> ; 理由: <msg> . |
| FailedUpdateStatus | 状況の更新に失敗しました。 |

表8.11 ネットワークイベント (openshift-sdn)

| 名前 | 説明 |
|-----------------|----------------------|
| Starting | OpenShiftSDN を開始します。 |

| 名前 | 説明 |
|----------------------|------------------------------------|
| NetworkFailed | Podのネットワークインターフェイスがなくなり、Podが停止します。 |

表8.12 ネットワークイベント (kube-proxy)

| 名前 | 説明 |
|-----------------|--|
| NeedPods | サービスポート <serviceName>:<port> は Pod が必要です。 |

表8.13 ボリュームイベント

| 名前 | 説明 |
|----------------------------------|---|
| FailedBinding | 利用可能な永続ボリュームがなく、ストレージクラスが設定されていません。 |
| VolumeMismatch | ボリュームサイズまたはクラスが要求の内容と異なります。 |
| VolumeFailedRecycle | 再利用 Pod の作成エラー |
| VolumeRecycled | ボリュームの再利用時に発生します。 |
| RecyclerPod | Pod の再利用時に発生します。 |
| VolumeDelete | ボリュームの削除時に発生します。 |
| VolumeFailedDelete | ボリュームの削除時のエラー。 |
| ExternalProvisioning | 要求のボリュームが手動または外部ソフトウェアでプロビジョニングされる場合に発生します。 |
| ProvisioningFailed | ボリュームのプロビジョニングに失敗しました。 |
| ProvisioningCleanupFailed | プロビジョニングしたボリュームの消去エラー |
| ProvisioningSucceeded | ボリュームが正常にプロビジョニングされる場合に発生します。 |
| WaitForFirstConsumer | Pod のスケジューリングまでバインドが遅延します。 |

表8.14 ライフサイクルフック

| 名前 | 説明 |
|------------------------------|--------------------------|
| FailedPostStart Hook | ハンドラーが Pod の起動に失敗しました。 |
| FailedPreStopHook | ハンドラーが pre-stop に失敗しました。 |
| UnfinishedPreStopHook | Pre-stop フックが完了しませんでした。 |

表8.15 デプロイメント

| 名前 | 説明 |
|-------------------------------------|-------------------------------|
| DeploymentCancellationFailed | デプロイメントのキャンセルに失敗しました。 |
| DeploymentCancelled | デプロイメントがキャンセルされました。 |
| DeploymentCreated | 新規レプリケーションコントローラーが作成されました。 |
| IngressIPRangeFull | サービスに割り当てる Ingress IP がありません。 |

表8.16 スケジューラーイベント

| 名前 | 説明 |
|-------------------------|---|
| FailedScheduling | Pod のスケジューリングに失敗: <pod-namespace>/<pod-name> 。このイベントは AssumePodVolumes の失敗、バインドの拒否など、複数の理由で発生します。 |
| Preempted | ノード <node-name> にある <preemptor-namespace>/<preemptor-name> |
| Scheduled | <node-name> に <pod-name> が正常に割り当てられました。 |

表8.17 デーモンセットイベント

| 名前 | 説明 |
|------------------------|--|
| SelectingAll | この DaemonSet は全 Pod を選択しています。空でないセレクターが必要です。 |
| FailedPlacement | <node-name> への Pod の配置に失敗しました。 |

| 名前 | 説明 |
|------------------------|--|
| FailedDaemonPod | ノード <node-name> で問題のあるデーモン Pod <pod-name> が見つかりました。この Pod の終了を試行します。 |

表8.18 LoadBalancer サービスイベント

| 名前 | 説明 |
|-----------------------------------|---|
| CreatingLoadBalancerFailed | ロードバランサーの作成エラー |
| DeletingLoadBalancer | ロードバランサーを削除します。 |
| EnsuringLoadBalancer | ロードバランサーを確保します。 |
| EnsuredLoadBalancer | ロードバランサーを確保しました。 |
| UnavailableLoadBalancer | LoadBalancer サービスに利用可能なノードがありません。 |
| LoadBalancerSourceRanges | 新規の LoadBalancerSourceRanges を表示します。例: <old-source-range> → <new-source-range> |
| LoadbalancerIP | 新しい IP アドレスを表示します。例: <old-ip> → <new-ip> |
| ExternalIP | 外部 IP アドレスを表示します。例: Added: <external-ip> |
| UID | 新しい UID を表示します。例: <old-service-uid> → <new-service-uid> |
| ExternalTrafficPolicy | 新しい ExternalTrafficPolicy を表示します。例: <old-policy> → <new-policy> |
| HealthCheckNodePort | 新しい HealthCheckNodePort を表示します。例: <old-node-port> → new-node-port |
| UpdatedLoadBalancer | 新規ホストでロードバランサーを更新しました。 |
| LoadBalancerUpdateFailed | 新規ホストでのロードバランサーの更新に失敗しました。 |
| DeletingLoadBalancer | ロードバランサーを削除します。 |

| 名前 | 説明 |
|-----------------------------------|------------------|
| DeletingLoadBalancerFailed | ロードバランサーの削除エラー。 |
| DeletedLoadBalancer | ロードバランサーを削除しました。 |

8.2. RED HAT OPENSIFT SERVICE ON AWS のノードが保持できる POD の数の見積り

クラスター管理者は、**OpenShift Cluster Capacity Tool** を使用して、現在のリソースが使い切られる前にそれらを増やすべくスケジュール可能な Pod 数を表示し、スケジュール可能な Pod 数を表示したり、Pod を今後スケジュールできるようにすることができます。この容量は、クラスター内の個別ノードからのものを集めたものであり、これには CPU、メモリー、ディスク領域などが含まれます。

8.2.1. OpenShift Cluster Capacity Tool について

OpenShift Cluster Capacity Tool は、より正確な見積もりを出すべく、スケジュールの一連の意思決定をシミュレーションし、リソースが使い切られる前にクラスターでスケジュールできる入力 Pod のインスタンス数を判別します。

注記

ノード間に分散しているすべてのリソースがカウントされないため、残りの割り当て可能な容量は概算となります。残りのリソースのみが分析対象となり、クラスターでのスケジュール可能な所定要件を持つ Pod のインスタンス数という点から消費可能な容量を見積もります。

Pod のスケジューリングはその選択およびアフィニティー条件に基づいて特定のノードセットについてのみサポートされる可能性があります。そのため、クラスターでスケジュール可能な残りの Pod 数を見積もることが困難になる場合があります。

OpenShift Cluster Capacity Tool は、コマンドラインからスタンドアロンのユーティリティーとして実行することも、Red Hat OpenShift Service on AWS クラスター内の Pod でジョブとして実行することもできます。これを Pod 内のジョブとしてツールを実行すると、介入なしに複数回実行することができます。

8.2.2. コマンドラインでの OpenShift Cluster Capacity Tool の実行

コマンドラインからOpenShift Cluster Capacity Tool を実行して、クラスターにスケジュール設定可能な Pod 数を見積ることができます。

ツールがリソース使用状況を見積もるために使用するサンプル Pod 仕様ファイルを作成します。pod spec はそのリソース要件を `limits` または `requests` として指定します。クラスター容量ツールは、Pod のリソース要件をその見積もりの分析に反映します。

前提条件

1. [OpenShift Cluster Capacity Tool](#) を実行します。これは、RedHat エコシステムカタログからコンテナイメージとして入手できます。
2. サンプルの Pod 仕様ファイルを作成します。
 - a. 以下のような YAML ファイルを作成します。

```
apiVersion: v1
kind: Pod
metadata:
  name: small-pod
  labels:
    app: guestbook
    tier: frontend
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
  - name: php-redis
    image: gcr.io/google-samples/gb-frontend:v4
    imagePullPolicy: Always
    resources:
      limits:
        cpu: 150m
        memory: 100Mi
      requests:
        cpu: 150m
        memory: 100Mi
  securityContext:
    allowPrivilegeEscalation: false
  capabilities:
    drop: [ALL]
```

- b. クラスターロールを作成します。

```
$ oc create -f <file_name>.yaml
```

以下に例を示します。

```
$ oc create -f pod-spec.yaml
```

手順

コマンドラインでクラスター容量ツールを使用するには、次のようにします。

1. ターミナルから、RedHat レジストリーにログインします。

```
$ podman login registry.redhat.io
```

2. クラスター容量ツールのイメージをプルします。

```
$ podman pull registry.redhat.io/openshift4/ose-cluster-capacity
```

3. クラスター容量ツールを実行します。

```
$ podman run -v $HOME/.kube:/kube:Z -v $(pwd):/cc:Z ose-cluster-capacity \
/bin/cluster-capacity --kubeconfig /kube/config --<pod_spec>.yaml \
/cc/<pod_spec>.yaml \
--verbose
```

ここでは、以下のようになります。

<pod_spec>.yaml

使用する Pod の仕様を指定します。

verbose

クラスター内の各ノードでスケジュールできる Pod の数の詳細な説明を出力します。

出力例

small-pod pod requirements:

- CPU: 150m
- Memory: 100Mi

The cluster can schedule 88 instance(s) of the pod small-pod.

Termination reason: Unschedulable: 0/5 nodes are available: 2 Insufficient cpu, 3 node(s) had taint {node-role.kubernetes.io/master: }, that the pod didn't tolerate.

Pod distribution among nodes:**small-pod**

- 192.168.124.214: 45 instance(s)
- 192.168.124.120: 43 instance(s)

上記の例では、クラスターにスケジュールできる推定 Pod の数は 88 です。

8.2.3. OpenShift Cluster Capacity Tool を Pod 内のジョブとして実行する

OpenShift Cluster Capacity Tool を Pod 内のジョブとして実行すると、ユーザーの介入を必要とせずにツールを複数回実行できます。OpenShift Cluster Capacity Tool は、ConfigMap オブジェクトを使用してジョブとして実行します。

前提条件

[OpenShift Cluster Capacity Tool](#) をダウンロードしてインストールします。

手順

クラスター容量ツールを実行するには、以下の手順を実行します。

1. クラスターロールを作成します。
 - a. 以下のような YAML ファイルを作成します。

```
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: cluster-capacity-role
rules:
```

```

- apiGroups: [""]
  resources: ["pods", "nodes", "persistentvolumeclaims", "persistentvolumes",
"services", "replicationcontrollers"]
  verbs: ["get", "watch", "list"]
- apiGroups: ["apps"]
  resources: ["replicasets", "statefulsets"]
  verbs: ["get", "watch", "list"]
- apiGroups: ["policy"]
  resources: ["poddisruptionbudgets"]
  verbs: ["get", "watch", "list"]
- apiGroups: ["storage.k8s.io"]
  resources: ["storageclasses"]
  verbs: ["get", "watch", "list"]

```

b.

次のコマンドを実行して、クラスターのロールを作成します。

```
$ oc create -f <file_name>.yaml
```

以下に例を示します。

```
$ oc create sa cluster-capacity-sa
```

2.

サービスアカウントを作成します。

```
$ oc create sa cluster-capacity-sa -n default
```

3.

ロールをサービスアカウントに追加します。

```
$ oc adm policy add-cluster-role-to-user cluster-capacity-role \
system:serviceaccount:<namespace>:cluster-capacity-sa
```

ここでは、以下のようになります。

<namespace>

Pod が配置されている名前空間を指定します。

4.

Pod 仕様を定義して、作成します。

a.

以下のような YAML ファイルを作成します。

```
apiVersion: v1
kind: Pod
metadata:
  name: small-pod
  labels:
    app: guestbook
    tier: frontend
spec:
  securityContext:
    runAsNonRoot: true
  seccompProfile:
    type: RuntimeDefault
  containers:
  - name: php-redis
    image: gcr.io/google-samples/gb-frontend:v4
    imagePullPolicy: Always
  resources:
    limits:
      cpu: 150m
      memory: 100Mi
    requests:
      cpu: 150m
      memory: 100Mi
  securityContext:
    allowPrivilegeEscalation: false
  capabilities:
    drop: [ALL]
```

b.

以下のコマンドを実行して Pod を作成します。

```
$ oc create -f <file_name>.yaml
```

以下に例を示します。

```
$ oc create -f pod.yaml
```

5.

以下のコマンドを実行して config map オブジェクトを作成します。

```
$ oc create configmap cluster-capacity-configmap \
  --from-file=pod.yaml=pod.yaml
```

クラスター容量分析は、cluster-capacity-configmap という名前の ConfigMap オブジェクトを使用してボリュームにマウントされ、入力 Pod 仕様ファイル pod.yaml はパス /test-

pod のボリューム `test-volume` にマウントされます。

6.

ジョブ仕様ファイルの以下のサンプルを使用して、ジョブを作成します。

a.

以下のような YAML ファイルを作成します。

```
apiVersion: batch/v1
kind: Job
metadata:
  name: cluster-capacity-job
spec:
  parallelism: 1
  completions: 1
  template:
    metadata:
      name: cluster-capacity-pod
    spec:
      containers:
      - name: cluster-capacity
        image: openshift/origin-cluster-capacity
        imagePullPolicy: "Always"
        volumeMounts:
        - mountPath: /test-pod
          name: test-volume
      env:
      - name: CC_INCLUSTER 1
        value: "true"
      command:
      - "/bin/sh"
      - "-ec"
      - |
        /bin/cluster-capacity --podspec=/test-pod/pod.yaml --verbose
      restartPolicy: "Never"
      serviceAccountName: cluster-capacity-sa
      volumes:
      - name: test-volume
        configMap:
          name: cluster-capacity-configmap
```

1

クラスター容量ツールにクラスター内で Pod として実行されていることを認識させる環境変数です。

ConfigMap の `pod.yaml` キーは Pod 仕様ファイル名と同じですが、これは必須ではありません。これを実行することで、入力 Pod 仕様ファイルは `/test-pod/pod.yaml` として Pod 内でアクセスできます。

b.

次のコマンドを実行して、クラスター容量イメージを Pod 内のジョブとして実行し

ます。

```
$ oc create -f cluster-capacity-job.yaml
```

検証

1.

ジョブログを確認し、クラスター内でスケジュールできる Pod の数を確認します。

```
$ oc logs jobs/cluster-capacity-job
```

出力例

```
small-pod pod requirements:
```

- CPU: 150m
- Memory: 100Mi

```
The cluster can schedule 52 instance(s) of the pod small-pod.
```

```
Termination reason: Unschedulable: No nodes are available that match all of the following predicates:: Insufficient cpu (2).
```

```
Pod distribution among nodes:
```

```
small-pod
```

- 192.168.124.214: 26 instance(s)
- 192.168.124.120: 26 instance(s)

8.3. 制限範囲によるリソース消費の制限

デフォルトで、コンテナは Red Hat OpenShift Service on AWS クラスターのバインドされていないコンピュートリソースで実行されます。制限範囲については、プロジェクト内の特定オブジェクトのリソースの消費を制限できます。

- **Pod およびコンテナ:** Pod およびそれらのコンテナの CPU およびメモリの最小および最大要件を設定できます。
- **イメージストリーム:** ImageStream オブジェクトのイメージおよびタグの数に制限を設定できます。

- イメージ: 内部レジストリーにプッシュできるイメージのサイズを制限することができます。
- 永続ボリューム要求 (PVC): 要求できる PVC のサイズを制限できます。

Pod が制限範囲で課される制約を満たさない場合、Pod を namespace に作成することはできません。

8.3.1. 制限範囲について

LimitRange オブジェクトで定義される制限範囲。プロジェクトのリソース消費を制限します。プロジェクトで、Pod、コンテナ、イメージ、イメージストリーム、または永続ボリューム要求 (PVC) の特定のリソース制限を設定できます。

すべてのリソース作成および変更要求は、プロジェクトのそれぞれの LimitRange オブジェクトに対して評価されます。リソースが列挙される制約のいずれかに違反する場合、そのリソースは拒否されます。

以下は、Pod、コンテナ、イメージ、イメージストリーム、または PVC のすべてのコンポーネントの制限範囲オブジェクトを示しています。同じオブジェクト内のこれらのコンポーネントのいずれかまたはすべての制限を設定できます。リソースを制御するプロジェクトごとに、異なる制限範囲オブジェクトを作成します。

コンテナの制限オブジェクトのサンプル

```
apiVersion: "v1"
kind: "LimitRange"
metadata:
  name: "resource-limits"
spec:
  limits:
    - type: "Container"
      max:
        cpu: "2"
        memory: "1Gi"
      min:
        cpu: "100m"
        memory: "4Mi"
      default:
        cpu: "300m"
        memory: "200Mi"
```

```
defaultRequest:  
  cpu: "200m"  
  memory: "100Mi"  
maxLimitRequestRatio:  
  cpu: "10"
```

8.3.1.1. コンポーネントの制限について

以下の例は、それぞれのコンポーネントの制限範囲パラメーターを示しています。これらの例は明確にするために使用されます。必要に応じて、いずれかまたはすべてのコンポーネントの単一の `LimitRange` オブジェクトを作成できます。

8.3.1.1.1. コンテナの制限

制限範囲により、Pod の各コンテナが特定のプロジェクトについて要求できる最小および最大 CPU およびメモリーを指定できます。コンテナがプロジェクトに作成される場合、Pod 仕様のコンテナ CPU およびメモリー要求は `LimitRange` オブジェクトに設定される値に準拠する必要があります。そうでない場合には、Pod は作成されません。

- コンテナの CPU またはメモリーの要求および制限は、`LimitRange` オブジェクトで指定されるコンテナの `min` リソース制約以上である必要があります。
 - コンテナの CPU またはメモリーの要求と制限は、`LimitRange` オブジェクトで指定されたコンテナの `max` リソース制約以下である必要があります。
- `LimitRange` オブジェクトが `max CPU` を定義する場合、Pod 仕様に `CPU request` 値を定義する必要はありません。ただし、制限範囲で指定される最大 CPU 制約を満たす `CPU limit` 値を指定する必要があります。
- コンテナ制限の要求に対する比率は、`LimitRange` オブジェクトに指定されるコンテナの `maxLimitRequestRatio` 値以下である必要があります。

`LimitRange` オブジェクトで `maxLimitRequestRatio` 制約を定義する場合、新規コンテナには `request` および `limit` 値の両方が必要になります。Red Hat OpenShift Service on AWS は、`limit` を `request` で除算して制限の要求に対する比率を算出します。この値は、1 より大きい正の整数でなければなりません。

たとえば、コンテナの `limit` 値が `cpu: 500` で、`request` 値が `cpu: 100` である場合、`cpu` の要求に対する制限の比は 5 になります。この比率は `maxLimitRequestRatio` より小

さいか等しくなければなりません。

Pod 仕様でコンテナリソースメモリーまたは制限を指定しない場合、制限範囲オブジェクトに指定されるコンテナの `default` または `defaultRequest CPU` およびメモリー値はコンテナに割り当てられます。

コンテナ `LimitRange` オブジェクトの定義

```
apiVersion: "v1"
kind: "LimitRange"
metadata:
  name: "resource-limits" 1
spec:
  limits:
    - type: "Container"
      max:
        cpu: "2" 2
        memory: "1Gi" 3
      min:
        cpu: "100m" 4
        memory: "4Mi" 5
      default:
        cpu: "300m" 6
        memory: "200Mi" 7
      defaultRequest:
        cpu: "200m" 8
        memory: "100Mi" 9
      maxLimitRequestRatio:
        cpu: "10" 10
```

1

制限範囲オブジェクトの名前です。

2

Pod の単一コンテナが要求できる CPU の最大量です。

3

Pod の単一コンテナが要求できるメモリーの最大量です。

4

Pod の単一コンテナが要求できる CPU の最小量です。

5

Pod の単一コンテナが要求できるメモリーの最小量です。

6

コンテナが使用できる CPU のデフォルト量 (Pod 仕様に指定されていない場合)。

7

コンテナが使用できるメモリーのデフォルト量 (Pod 仕様に指定されていない場合)。

8

コンテナが要求できる CPU のデフォルト量 (Pod 仕様に指定されていない場合)。

9

コンテナが要求できるメモリーのデフォルト量 (Pod 仕様に指定されていない場合)。

10

コンテナの要求に対する制限の最大比率。

8.3.1.1.2. Pod の制限

制限範囲により、所定プロジェクトの Pod 全体でのすべてのコンテナの CPU およびメモリーの最小および最大の制限を指定できます。コンテナをプロジェクトに作成するには、Pod 仕様のコンテナ CPU およびメモリー要求は `LimitRange` オブジェクトに設定される値に準拠する必要があります。そうでない場合には、Pod は作成されません。

Pod 仕様でコンテナリソースメモリーまたは制限を指定しない場合、制限範囲オブジェクトに指定されるコンテナの `default` または `defaultRequest` CPU およびメモリー値はコンテナに割り当てられます。

Pod のすべてのコンテナにおいて、以下を満たしている必要があります。

•

コンテナの CPU またはメモリーの要求および制限は、`LimitRange` オブジェクトに指

定される Pod の min リソース制約以上である必要があります。

- コンテナの CPU またはメモリーの要求および制限は、LimitRange オブジェクトに指定される Pod の max リソース制約以下である必要があります。
- コンテナ制限の要求に対する比率は、LimitRange オブジェクトに指定される maxLimitRequestRatio 制約以下である必要があります。

Pod LimitRange オブジェクト定義

```
apiVersion: "v1"
kind: "LimitRange"
metadata:
  name: "resource-limits" ①
spec:
  limits:
    - type: "Pod"
      max:
        cpu: "2" ②
        memory: "1Gi" ③
      min:
        cpu: "200m" ④
        memory: "6Mi" ⑤
      maxLimitRequestRatio:
        cpu: "10" ⑥
```

①

制限範囲オブジェクトの名前です。

②

すべてのコンテナにおいて Pod が要求できる CPU の最大量です。

③

すべてのコンテナにおいて Pod が要求できるメモリーの最大量です。

④

すべてのコンテナにおいて Pod が要求できる CPU の最小量です。

5

すべてのコンテナにおいて Pod が要求できるメモリーの最小量です。

6

コンテナの要求に対する制限の最大比率。

8.3.1.1.3. イメージの制限

LimitRange オブジェクトを使用すると、OpenShift イメージレジストリーにプッシュできるイメージの最大サイズを指定できます。

OpenShift イメージレジストリーにイメージをプッシュする場合、以下を満たす必要があります。

- イメージのサイズは、LimitRange オブジェクトで指定されるイメージの max サイズ以下である必要があります。

イメージ LimitRange オブジェクトの定義

```
apiVersion: "v1"
kind: "LimitRange"
metadata:
  name: "resource-limits" ①
spec:
  limits:
    - type: openshift.io/Image
      max:
        storage: 1Gi ②
```

1

LimitRange オブジェクトの名前。

2

OpenShift イメージレジストリーにプッシュできるイメージの最大サイズ。



警告

イメージのサイズは、アップロードされるイメージのマニフェストで常に表示される訳ではありません。これは、とりわけ Docker 1.10 以上で作成され、v2 レジストリーにプッシュされたイメージの場合に該当します。このようなイメージが古い Docker デーモンでプルされると、イメージマニフェストはレジストリーによってスキーマ v1 に変換されますが、この場合サイズ情報が欠落します。イメージに設定されるストレージの制限がこのアップロードを防ぐことはありません。

現在、[この問題](#) への対応が行われています。

8.3.1.1.4. イメージストリームの制限

LimitRange オブジェクトにより、イメージストリームの制限を指定できます。

各イメージストリームについて、以下が当てはまります。

- ImageStream 仕様のイメージタグ数は、LimitRange オブジェクトの `openshift.io/image-tags` 制約以下である必要があります。
- ImageStream 仕様のイメージへの一意の参照数は、制限範囲オブジェクトの `openshift.io/images` 制約以下である必要があります。

イメージストリーム LimitRange オブジェクト定義

```
apiVersion: "v1"
kind: "LimitRange"
metadata:
  name: "resource-limits" 1
spec:
  limits:
```

```
- type: openshift.io/ImageStream
  max:
    openshift.io/image-tags: 20 ②
    openshift.io/images: 30 ③
```

①

LimitRange オブジェクトの名前。

②

イメージストリーム仕様の `imagestream.spec.tags` パラメーターの一意のイメージタグの最大数。

③

`imagestream` 仕様の `imagestream.status.tags` パラメーターの一意のイメージ参照の最大数。

`openshift.io/image-tags` リソースは、一意のイメージ参照を表します。使用できる参照は、`ImageStreamTag`、`ImageStreamImage` および `DockerImage` になります。タグは、`oc tag` および `oc import-image` コマンドを使用して作成できます。内部参照か外部参照であるかの区別はありません。ただし、`ImageStream` の仕様でタグ付けされる一意の参照はそれぞれ 1 回のみカウントされます。内部コンテナイメージレジストリーへのプッシュを制限しませんが、タグの制限に役立ちます。

`openshift.io/images` リソースは、イメージストリームのステータスに記録される一意のイメージ名を表します。これにより、`OpenShift` イメージレジストリーにプッシュできるイメージ数を制限できます。内部参照か外部参照であるかの区別はありません。

8.3.1.1.5. 永続ボリューム要求 (PVC) の制限

LimitRange オブジェクトにより、永続ボリューム要求 (PVC) で要求されるストレージを制限できます。

プロジェクトのすべての永続ボリューム要求 (PVC) において、以下が一致している必要があります。

- 永続ボリューム要求 (PVC) のリソース要求は、LimitRange オブジェクトに指定される PVC の min 制約以上である必要があります。

- 永続ボリューム要求 (PVC) のリソース要求は、LimitRange オブジェクトに指定される PVC の max 制約以下である必要があります。

PVC LimitRange オブジェクト定義

```
apiVersion: "v1"
kind: "LimitRange"
metadata:
  name: "resource-limits" ①
spec:
  limits:
    - type: "PersistentVolumeClaim"
      min:
        storage: "2Gi" ②
      max:
        storage: "50Gi" ③
```

①

LimitRange オブジェクトの名前。

②

永続ボリューム要求 (PVC) で要求できるストレージの最小量です。

③

永続ボリューム要求 (PVC) で要求できるストレージの最大量です。

8.3.2. 制限範囲の作成

制限範囲をプロジェクトに適用するには、以下を実行します。

1.

必要な仕様で LimitRange オブジェクトを作成します。

```
apiVersion: "v1"
```

```
kind: "LimitRange"
metadata:
  name: "resource-limits" ❶
spec:
  limits:
    - type: "Pod" ❷
      max:
        cpu: "2"
        memory: "1Gi"
      min:
        cpu: "200m"
        memory: "6Mi"
    - type: "Container" ❸
      max:
        cpu: "2"
        memory: "1Gi"
      min:
        cpu: "100m"
        memory: "4Mi"
      default: ❹
        cpu: "300m"
        memory: "200Mi"
      defaultRequest: ❺
        cpu: "200m"
        memory: "100Mi"
      maxLimitRequestRatio: ❻
        cpu: "10"
    - type: openshift.io/Image ❼
      max:
        storage: 1Gi
    - type: openshift.io/ImageStream ❽
      max:
        openshift.io/image-tags: 20
        openshift.io/images: 30
    - type: "PersistentVolumeClaim" ❾
      min:
        storage: "2Gi"
      max:
        storage: "50Gi"
```

❶

LimitRange オブジェクトの名前を指定します。

❷

Pod の制限を設定するには、必要に応じて CPU およびメモリーの最小および最大要求を指定します。

❸

コンテナの制限を設定するには、必要に応じて CPU およびメモリーの最小および最大要求を指定します。

4

オプション: コンテナの場合、Pod 仕様で指定されていない場合、コンテナが使用できる CPU またはメモリのデフォルト量を指定します。

5

オプション: コンテナの場合、Pod 仕様で指定されていない場合、コンテナが要求できる CPU またはメモリのデフォルト量を指定します。

6

オプション: コンテナの場合、Pod 仕様で指定できる要求に対する制限の最大比率を指定します。

7

Image オブジェクトに制限を設定するには、OpenShift イメージレジストリーにプッシュできるイメージの最大サイズを設定します。

8

イメージストリームの制限を設定するには、必要に応じて ImageStream オブジェクトファイルにあるイメージタグおよび参照の最大数を設定します。

9

永続ボリューム要求 (PVC) の制限を設定するには、要求できるストレージの最小および最大量を設定します。

2.

オブジェクトを作成します。

```
$ oc create -f <limit_range_file> -n <project> 1
```

1

作成した YAML ファイルの名前と、制限を適用する必要があるプロジェクトを指定します。

8.3.3. 制限の表示

Web コンソールでプロジェクトの Quota ページに移動し、プロジェクトで定義される制限を表示で

きます。

CLI を使用して制限範囲の詳細を表示することもできます。

1.

プロジェクトで定義される `LimitRange` オブジェクトのリストを取得します。たとえば、`demoproject` というプロジェクトの場合は以下のようになります。

```
$ oc get limits -n demoproject
```

```
NAME          CREATED AT
resource-limits 2020-07-15T17:14:23Z
```

2.

関連のある `LimitRange` オブジェクトを記述します。たとえば、`resource-limits` 制限範囲の場合は以下のようになります。

```
$ oc describe limits resource-limits -n demoproject
```

```
Name:          resource-limits
Namespace:     demoproject
Type          Resource      Min   Max   Default Request Default Limit
Max Limit/Request Ratio
-----
Pod          cpu            200m  2    -     -     -
Pod          memory        6Mi   1Gi  -     -     -
Container    cpu           100m  2    200m  300m  10
Container    memory       4Mi   1Gi  100Mi 200Mi  -
openshift.io/Image      storage      -     1Gi  -     -     -
openshift.io/ImageStream openshift.io/image -    12  -     -     -
openshift.io/ImageStream openshift.io/image-tags -   10  -     -     -
PersistentVolumeClaim   storage      -    50Gi -     -     -
```

8.3.4. 制限範囲の削除

プロジェクトで制限を実施しないように有効な `LimitRange` オブジェクト削除するには、以下を実行します。

-

以下のコマンドを実行します。

```
$ oc delete limits <limit_name>
```

8.4. コンテナメモリーとリスク要件を満たすためのクラスターメモリーの設定

クラスター管理者は、以下を実行し、クラスターがアプリケーションメモリーの管理を通じて効率的に動作するようにすることができます。

- コンテナ化されたアプリケーションコンポーネントのメモリーおよびリスク要件を判別し、それらの要件を満たすようコンテナメモリーパラメーターを設定する
- コンテナ化されたアプリケーションランタイム (OpenJDK など) を、設定されたコンテナメモリーパラメーターに基づいて最適に実行されるよう設定する
- コンテナでの実行に関連するメモリー関連のエラー状態を診断し、これを解決する

8.4.1. アプリケーションメモリーの管理について

まず Red Hat OpenShift Service on AWS によるコンピュートリソースの管理方法の概要をよく読んでから次の手順に進むことを推奨します。

各種のリソース (メモリー、cpu、ストレージ) に応じて、Red Hat OpenShift Service on AWS ではオプションの 要求 および 制限 の値を Pod の各コンテナに設定できます。

メモリー要求とメモリー制限について、以下の点に注意してください。

- **メモリー要求**
 - メモリー要求値が指定されている場合、Red Hat OpenShift Service on AWS スケジューラーに影響します。スケジューラーは、コンテナのノードへのスケジュール時にメモリー要求を考慮し、コンテナの使用のために選択されたノードで要求されたメモリーをフェンスオフします。
 - ノードのメモリーが使い切られると、Red Hat OpenShift Service on AWS はメモリー使用がメモリー要求を最も超過しているコンテナのエビクションを優先します。深刻なメモリー枯渇の場合、ノード OOM キラーが同様のメトリックに基づいてコンテナ内のプロセスを選択して強制終了することがあります。
 - クラスター管理者は、メモリー要求値に対してクォータを割り当てるか、デフォルト値を割り当てるすることができます。

- クラスター管理者は、クラスターのオーバーコミットを管理するために開発者が指定するメモリー要求の値を上書きできます。
- - メモリー制限
 - メモリー制限値が指定されている場合、コンテナのすべてのプロセスに割り当て可能なメモリーにハード制限を指定します。
 - コンテナのすべてのプロセスで割り当てられるメモリーがメモリー制限を超過する場合、ノードの OOM (Out of Memory) killer はコンテナのプロセスをすぐに選択し、これを強制終了します。
 - メモリー要求とメモリー制限の両方が指定される場合、メモリー制限の値はメモリー要求の値よりも大きいのか、これと等しくなければなりません。
 - クラスター管理者は、メモリーの制限値に対してクォータを割り当てるか、デフォルト値を割り当てることができます。
 - 最小メモリー制限は 12 MB です。Cannot allocate memory Pod イベントのためにコンテナの起動に失敗すると、メモリー制限は低くなります。メモリー制限を引き上げるか、これを削除します。制限を削除すると、Pod は制限のないノードのリソースを消費できるようになります。

8.4.1.1. アプリケーションメモリーストラテジーの管理

Red Hat OpenShift Service on AWS でアプリケーションメモリーをサイジングする手順は以下の通りです。

1. 予想されるコンテナのメモリー使用の判別

必要時に予想される平均およびピーク時のコンテナのメモリー使用を判別します (例: 別の負荷テストを実行)。コンテナで並行して実行されている可能性のあるすべてのプロセスを必ず考慮に入れるようにしてください。たとえば、メインのアプリケーションは付属スクリプトを生成しているかどうかを確認します。

2.

リスク選好 (risk appetite) の判別

エビクションのリスク選好を判別します。リスク選好のレベルが低い場合、コンテナは予想されるピーク時の使用量と安全マージンのパーセンテージに応じてメモリーを要求します。リスク選好が高くなる場合、予想される平均の使用量に応じてメモリーを要求することがより適切な場合があります。

3.

コンテナのメモリー要求の設定

上記に基づいてコンテナのメモリー要求を設定します。要求がアプリケーションのメモリー使用をより正確に表示することが望ましいと言えます。要求が高すぎる場合には、クラスターおよびクォータの使用が非効率となります。要求が低すぎる場合、アプリケーションのエビクションの可能性が高くなります。

4.

コンテナのメモリー制限の設定 (必要な場合)

必要時にコンテナのメモリー制限を設定します。制限を設定すると、コンテナのすべてのプロセスのメモリー使用量の合計が制限を超える場合にコンテナのプロセスがすぐに強制終了されるため、いくつかの利点をもたらします。まずは予期しないメモリー使用の超過を早期に明確にする (fail fast (早く失敗する)) ことができ、次にプロセスをすぐに中止できます。

一部の Red Hat OpenShift Service on AWS クラスターでは制限値を設定する必要があります。制限に基づいて要求を上書きする場合があります。また、一部のアプリケーションイメージは、要求値よりも検出が簡単なことから設定される制限値に依存します。

メモリー制限が設定される場合、これは予想されるピーク時のコンテナのメモリー使用量と安全マージンのパーセンテージよりも低い値に設定することはできません。

5.

アプリケーションが調整されていることの確認

適切な場合は、設定される要求および制限値に関連してアプリケーションが調整されていることを確認します。この手順は、JVM などのメモリーをプールするアプリケーションにおいてとくに当てはまります。残りの部分では、これについて説明します。

8.4.2. Red Hat OpenShift Service on AWS の OpenJDK 設定について

デフォルトの OpenJDK 設定はコンテナ化された環境では機能しません。そのため、コンテナで OpenJDK を実行する場合は常に追加の Java メモリー設定を指定する必要があります。

JVM のメモリーレイアウトは複雑で、バージョンに依存しており、本書ではこれについて詳細には説明しません。ただし、コンテナで OpenJDK を実行する際のスタートにあたって少なくとも以下の 3 つのメモリー関連のタスクが主なタスクになります。

1. JVM 最大ヒープサイズを上書きする。
2. JVM が未使用メモリーをオペレーティングシステムに解放するよう促す (適切な場合)。
3. コンテナ内のすべての JVM プロセスが適切に設定されていることを確認する。

コンテナでの実行に向けて JVM ワークロードを最適に調整する方法については本書では扱いませんが、これには複数の JVM オプションを追加で設定することが必要になる場合があります。

8.4.2.1. JVM の最大ヒープサイズを上書きする方法について

数多くの Java ワークロードにおいて、JVM ヒープはメモリーの最大かつ単一のコンシューマーです。現時点で OpenJDK は、OpenJDK がコンテナ内で実行されているかにかかわらず、ヒープに使用されるコンピュータノードのメモリーの最大 1/4 (1/-XX:MaxRAMFraction) を許可するようデフォルトで設定されます。そのため、コンテナのメモリー制限も設定されている場合には、この動作をオーバーライドすることが必須です。

上記を実行する方法として、2 つ以上の方法を使用できます：

- コンテナのメモリー制限が設定されており、JVM で実験的なオプションがサポートされている場合には、`-XX:+UnlockExperimentalVMOptions -XX:+UseCGroupMemoryLimitForHeap` を設定します。



注記

JDK 11 では `UseCGroupMemoryLimitForHeap` オプションが削除されました。`-XX:+UseContainerSupport` を代わりに使用します。

これにより、`-XX:MaxRAM` がコンテナのメモリー制限に設定され、最大ヒープサイズ (`-XX:MaxHeapSize / -Xmx`) が `1/-XX:MaxRAMFraction` に設定されます (デフォルトでは

1/4)。

- **-XX:MaxRAM、-XX:MaxHeapSize または -Xmx のいずれかを直接上書きします。**

このオプションには、値のハードコーディングが必要になりますが、安全マージンを計算できるという利点があります。

8.4.2.2. JVM で未使用メモリーをオペレーティングシステムに解放するよう促す方法について

デフォルトで、OpenJDK は未使用メモリーをオペレーティングシステムに積極的に返しませんが、これは多くのコンテナ化された Java ワークロードには適していますが、例外として、コンテナ内に JVM と共存する追加のアクティブなプロセスがあるワークロードの場合を考慮する必要があります。それらの追加のプロセスはネイティブのプロセスである場合や追加の JVM の場合、またはこれら 2 つの組み合わせである場合もあります。

Java ベースのエージェントは、次の JVM 引数を使用して、JVM が未使用のメモリーをオペレーティングシステムに解放するよう促すことができます。

```
-XX:+UseParallelGC
-XX:MinHeapFreeRatio=5 -XX:MaxHeapFreeRatio=10 -XX:GCTimeRatio=4
-XX:AdaptiveSizePolicyWeight=90.
```

これらの引数は、割り当てられたメモリーが使用中のメモリー (-XX:MaxHeapFreeRatio) の 110% を超え、ガベージコレクター (-XX:GCTimeRatio) での CPU 時間の 20% を使用する場合は常にヒープメモリーをオペレーティングシステムに返すことが意図されています。アプリケーションのヒープ割り当てが初期のヒープ割り当て (-XX:InitialHeapSize / -Xms で上書きされる) を下回ることはありません。詳細情報については、[Tuning Java's footprint in OpenShift \(Part 1\)](#)、[Tuning Java's footprint in OpenShift \(Part 2\)](#)、および [OpenJDK and Containers](#) を参照してください。

8.4.2.3. コンテナ内のすべての JVM プロセスが適切に設定されていることを確認する方法について

複数の JVM が同じコンテナで実行される場合、それらすべてが適切に設定されていることを確認する必要があります。多くのワークロードでは、それぞれの JVM に memory budget のパーセンテージを付与する必要があります。これにより大きな安全マージンが残される場合があります。

多くの Java ツールは JVM を設定するために各種の異なる環境変数 (JAVA_OPTS、GRADLE_OPTS など) を使用します。適切な設定が適切な JVM に渡されていることを確認するのが容易でない場合もあります。

JAVA_TOOL_OPTIONS 環境変数は常に OpenJDK によって考慮され、**JAVA_TOOL_OPTIONS**

に指定された値は、JVM コマンドラインに指定される他のオプションによって上書きされます。デフォルトでは、Java ベースのエージェントイメージで実行されるすべての JVM ワークロードに対してこれらのオプションがデフォルトで使用されるように、Red Hat OpenShift Service on AWS Jenkins Maven エージェントイメージを次のように設定します。

```
JAVA_TOOL_OPTIONS="-XX:+UnlockExperimentalVMOptions
-XX:+UseCGroupMemoryLimitForHeap -Dsun.zip.disableMemoryMapping=true"
```



注記

JDK 11 では UseCGroupMemoryLimitForHeap オプションが削除されました。-XX:+UseContainerSupport を代わりに使用します。

この設定は、追加オプションが要求されないことを保証する訳ではなく、有用な開始点になることを意図しています。

8.4.3. Pod 内でのメモリー要求および制限の検索

Pod 内からメモリー要求および制限を動的に検出するアプリケーションでは Downward API を使用する必要があります。

手順

1. MEMORY_REQUEST と MEMORY_LIMIT スタンザを追加するように Pod を設定します。
 - a. 以下のような YAML ファイルを作成します。

```
apiVersion: v1
kind: Pod
metadata:
  name: test
spec:
  securityContext:
    runAsNonRoot: true
  seccompProfile:
    type: RuntimeDefault
  containers:
  - name: test
    image: fedora:latest
    command:
    - sleep
    - "3600"
```

```
env:
- name: MEMORY_REQUEST ①
  valueFrom:
    resourceFieldRef:
      containerName: test
      resource: requests.memory
- name: MEMORY_LIMIT ②
  valueFrom:
    resourceFieldRef:
      containerName: test
      resource: limits.memory
resources:
  requests:
    memory: 384Mi
  limits:
    memory: 512Mi
securityContext:
  allowPrivilegeEscalation: false
capabilities:
  drop: [ALL]
```

①

このスタンプを追加して、アプリケーションメモリの要求値を見つけます。

②

このスタンプを追加して、アプリケーションメモリの制限値を見つけます。

b.

以下のコマンドを実行して Pod を作成します。

```
$ oc create -f <file-name>.yaml
```

検証

1.

リモートシェルを使用して Pod にアクセスします。

```
$ oc rsh test
```

2.

要求された値が適用されていることを確認します。

```
$ env | grep MEMORY | sort
```

出力例


```
MEMORY_LIMIT=536870912
MEMORY_REQUEST=402653184
```



注記

メモリー制限値は、`/sys/fs/cgroup/memory/memory.limit_in_bytes` ファイルによってコンテナ内から読み取ることもできます。

8.4.4. OOM の強制終了ポリシーについて

Red Hat OpenShift Service on AWS は、コンテナのすべてのプロセスのメモリー使用量の合計がメモリー制限を超えるか、またはノードのメモリーを使い切られるなどの深刻な状態が生じる場合にコンテナのプロセスを強制終了する場合があります。

プロセスが OOM (Out of Memory) によって強制終了される場合、コンテナがすぐに終了する場合があります。コンテナの PID 1 プロセスが SIGKILL を受信する場合、コンテナはすぐに終了します。それ以外の場合、コンテナの動作は他のプロセスの動作に依存します。

たとえば、コンテナのプロセスは、SIGKILL シグナルを受信したことを示すコード 137 で終了します。

コンテナがすぐに終了しない場合、OOM による強制終了は以下のように検出できます。

1. リモートシェルを使用して Pod にアクセスします。

```
# oc rsh test
```

2. 以下のコマンドを実行して、`/sys/fs/cgroup/memory/memory.oom_control` で現在の OOM kill カウントを表示します。

```
$ grep '^oom_kill' /sys/fs/cgroup/memory/memory.oom_control
```

出力例

```
oom_kill 0
```

3.

以下のコマンドを実行して、**Out Of Memory (OOM)** による強制終了を促します。

```
$ sed -e " </dev/zero
```

出力例

```
Killed
```

4.

以下のコマンドを実行して、**sed** コマンドの終了ステータスを表示します。

```
$ echo $?
```

出力例

```
137
```

137 コードは、コンテナのプロセスが、**SIGKILL** シグナルを受信したことを示すコード 137 で終了していることを示唆します。

5.

以下のコマンドを実行して、`/sys/fs/cgroup/memory/memory.oom_control` の **OOM kill** カウンターの増分を表示します。

```
$ grep '^oom_kill' /sys/fs/cgroup/memory/memory.oom_control
```

出力例

```
oom_kill 1
```

Pod の 1 つ以上のプロセスが OOM で強制終了され、Pod がこれに続いて終了する場合 (即時であるかどうかは問わない)、フェーズは Failed、理由は OOMKilled になります。OOM で強制終了された Pod は restartPolicy の値によって再起動する場合があります。再起動されない場合は、レプリケーションコントローラーなどのコントローラーが Pod の失敗したステータスを認識し、古い Pod に置き換わる新規 Pod を作成します。

以下のコマンドを使用して Pod のステータスを取得します。

```
$ oc get pod test
```

出力例

```
NAME      READY   STATUS    RESTARTS   AGE
test      0/1     OOMKilled 0           1m
```

-

Pod が再起動されていない場合は、以下のコマンドを実行して Pod を表示します。

```
$ oc get pod test -o yaml
```

出力例

```
...
status:
  containerStatuses:
  - name: test
    ready: false
    restartCount: 0
    state:
```

```
terminated:  
  exitCode: 137  
  reason: OOMKilled  
phase: Failed
```

- 再起動した場合は、以下のコマンドを実行して Pod を表示します。

```
$ oc get pod test -o yaml
```

出力例

```
...  
status:  
  containerStatuses:  
  - name: test  
    ready: true  
    restartCount: 1  
    lastState:  
      terminated:  
        exitCode: 137  
        reason: OOMKilled  
    state:  
      running:  
        phase: Running
```

8.4.5. Pod エビクションについて

Red Hat OpenShift Service on AWS は、ノードのメモリーが使い切られると、そのノードから Pod をエビクトする場合があります。メモリー消費の度合いによって、エビクションは正常に行われる場合もあれば、そうでない場合もあります。正常なエビクションは、各コンテナのメインプロセス (PID 1) が SIGTERM シグナルを受信してから、プロセスがすでに終了していない場合は後になって SIGKILL シグナルを受信することを意味します。正常ではないエビクションは各コンテナのメインプロセスが SIGKILL シグナルを即時に受信することを示します。

エビクトされた Pod のフェーズは Failed になり、理由は Evicted になります。この場合、restartPolicy の値に関係なく再起動されません。ただし、レプリケーションコントローラーなどのコントローラーは Pod の失敗したステータスを認識し、古い Pod に置き換わる新規 Pod を作成します。

```
$ oc get pod test
```

出力例

```
NAME    READY   STATUS    RESTARTS  AGE
test    0/1     Evicted  0          1m
```

```
$ oc get pod test -o yaml
```

出力例

```
...
status:
  message: 'Pod The node was low on resource: [MemoryPressure].'
```

```
  phase: Failed
```

```
  reason: Evicted
```

8.5. オーバーコミットされたノード上に POD を配置するためのクラスターの設定

オーバーコミットとは、コンテナの計算リソース要求と制限の合計が、そのシステムで利用できるリソースを超えた状態のことです。オーバーコミットの使用は、容量に対して保証されたパフォーマンスのトレードオフが許容可能である開発環境において必要になる場合があります。

コンテナは、コンピュータリソース要求および制限を指定することができます。要求はコンテナのスケジューリングに使用され、最小限のサービス保証を提供します。制限は、ノード上で消費できるコンピュータリソースの量を制限します。

スケジューラーは、クラスター内のすべてのノードにおけるコンピュータリソース使用の最適化を試行します。これは Pod のコンピュータリソース要求とノードの利用可能な容量を考慮に入れて Pod を特定のノードに配置します。

Red Hat OpenShift Service on AWS 管理者は、オーバーコミットのレベルを制御し、ノード上のコンテナの密度を管理できるようになりました。クラスターレベルのオーバーコミットを

ClusterResourceOverride Operator を使用して設定し、開発者用のコンテナに設定された要求と制限の比率について上書きすることができます。[ノードのオーバーコミット](#) と組み合わせて、リソースの制限と要求を調整して、必要なレベルのオーバーコミットを実現できます。



注記

Red Hat OpenShift Service on AWS では、クラスターレベルのオーバーコミットを有効にする必要があります。ノードのオーバーコミットはデフォルトで有効にされています。[ノードのオーバーコミットの無効化](#) を参照してください。

8.5.1. リソース要求とオーバーコミット

各コンピュータリソースについて、コンテナはリソース要求および制限を指定できます。スケジューリングの決定は要求に基づいて行われ、ノードに要求される値を満たす十分な容量があることが確認されます。コンテナが制限を指定するものの、要求を省略する場合、要求はデフォルトで制限値に設定されます。コンテナは、ノードの指定される制限を超えることはできません。

制限の実施方法は、コンピュータリソースのタイプによって異なります。コンテナが要求または制限を指定しない場合、コンテナはリソース保証のない状態でノードにスケジュールされます。実際に、コンテナはローカルの最も低い優先順位で利用できる指定リソースを消費できます。リソースが不足する状態では、リソース要求を指定しないコンテナに最低レベルの QoS (Quality of Service) が設定されます。

スケジューリングは要求されるリソースに基づいて行われる一方で、クォータおよびハード制限はリソース制限のことを指しており、これは要求されるリソースよりも高い値に設定できます。要求と制限の間の差異は、オーバーコミットのレベルを定めるものとなります。たとえば、コンテナに 1Gi のメモリー要求と 2Gi のメモリー制限が指定される場合、コンテナのスケジューリングはノードで 1Gi を利用可能とする要求に基づいて行われますが、2Gi まで使用することができます。そのため、この場合のオーバーコミットは 200% になります。

8.5.2. Cluster Resource Override Operator を使用したクラスターレベルのオーバーコミット

Cluster Resource Override Operator は、クラスター内のすべてのノードでオーバーコミットのレベルを制御し、コンテナの密度を管理できる受付 Webhook です。Operator は、特定のプロジェクトのノードが定義されたメモリーおよび CPU 制限を超える場合について制御します。

Cluster Resource Override Operator は、次のセクションに示すように、Red Hat OpenShift Service on AWS コンソールまたは CLI を使用してインストールする必要があります。インストール時に、以下の例のように、オーバーコミットのレベルを設定する ClusterResourceOverride カスタムリソース (CR) を作成します。

apiVersion: operator.autoscaling.openshift.io/v1

```

kind: ClusterResourceOverride
metadata:
  name: cluster ❶
spec:
  podResourceOverride:
    spec:
      memoryRequestToLimitPercent: 50 ❷
      cpuRequestToLimitPercent: 25 ❸
      limitCPUToMemoryPercent: 200 ❹
# ...

```

❶

名前は `cluster` でなければなりません。

❷

オプション: コンテナのメモリー制限が指定されているか、デフォルトに設定されている場合、メモリー要求は制限のパーセンテージ (1-100) に対して上書きされます。デフォルトは 50 です。

❸

オプション: コンテナの CPU 制限が指定されているか、デフォルトに設定されている場合、CPU 要求は、1-100 までの制限のパーセンテージに対応して上書きされます。デフォルトは 25 です。

❹

オプション: コンテナのメモリー制限が指定されているか、デフォルトに設定されている場合、CPU 制限は、指定されている場合にメモリーのパーセンテージに対して上書きされます。1Gi の RAM の 100 パーセントでのスケーリングは、1 CPU コアに等しくなります。これは、CPU 要求を上書きする前に処理されます (設定されている場合)。デフォルトは 200 です。



注記

Cluster Resource Override Operator の上書きは、制限がコンテナに設定されていない場合は影響を与えません。個別プロジェクトごとのデフォルト制限を使用して `LimitRange` オブジェクトを作成するか、`Pod` 仕様で制限を設定し、上書きが適用されるようにします。

設定時に、以下のラベルを各プロジェクトの `namespace` オブジェクトに適用し、上書きをプロジェクトごとに有効にできます。

```

apiVersion: v1
kind: Namespace

```

```
metadata:  
  
# ...  
  
labels:  
  clusterresourceoverrides.admission.autoscaling.openshift.io/enabled: "true"  
  
# ...
```

Operator は ClusterResourceOverride CR の有無を監視し、ClusterResourceOverride 受付 Webhook が Operator と同じ namespace にインストールされるようにします。

8.5.2.1. Web コンソールを使用した Cluster Resource Override Operator のインストール

Red Hat OpenShift Service on AWS の Web コンソールを使用して Cluster Resource Override Operator をインストールし、クラスターでオーバーコミットを制御できます。

前提条件

- 制限がコンテナに設定されていない場合、Cluster Resource Override Operator は影響を与えません。LimitRange オブジェクトを使用してプロジェクトのデフォルト制限を指定するか、Pod 仕様で制限を設定して上書きが適用されるようにする必要があります。

手順

Red Hat OpenShift Service on AWS の Web コンソールを使用して Cluster Resource Override Operator をインストールするには、以下を実行します。

1. Red Hat OpenShift Service on AWS の Web コンソールで、Home → Projects に移動します。
 - a. Create Project をクリックします。
 - b. clusterresourceoverride-operator をプロジェクトの名前として指定します。
 - c. Create をクリックします。
2. Operators → OperatorHub に移動します。

- a. 利用可能な Operator のリストから **ClusterResourceOverride Operator** を選択し、**Install** をクリックします。
 - b. **Install Operator** ページで、**A specific Namespace on the cluster** が **Installation Mode** について選択されていることを確認します。
 - c. **clusterresourceoverride-operator** が **Installed Namespace** について選択されていることを確認します。
 - d. **Update Channel** および **Approval Strategy** を選択します。
 - e. **Install** をクリックします。
3. **Installed Operators** ページで、**ClusterResourceOverride** をクリックします。
 - a. **ClusterResourceOverride Operator** 詳細ページで、**Create ClusterResourceOverride** をクリックします。
 - b. **Create ClusterResourceOverride** ページで、**YAML view** をクリックして、**YAML テンプレート**を編集し、必要に応じてオーバーコミット値を設定します。

```
apiVersion: operator.autoscaling.openshift.io/v1
kind: ClusterResourceOverride
metadata:
  name: cluster ①
spec:
  podResourceOverride:
    spec:
      memoryRequestToLimitPercent: 50 ②
      cpuRequestToLimitPercent: 25 ③
      limitCPUToMemoryPercent: 200 ④
# ...
```

①

名前は **cluster** でなければなりません。

②

3

オプション: コンテナ CPU の制限を上書きするためのパーセンテージが使用される場合は、これを 1-100 までの値で指定します。デフォルトは 25 です。

4

オプション: コンテナメモリーの制限を上書きするためのパーセンテージが使用される場合は、これを指定します。1Gi の RAM の 100 パーセントでのスケールアップは、1 CPU コアに等しくなります。これは、CPU 要求を上書きする前に処理されます (設定されている場合)。デフォルトは 200 です。

c.

Create をクリックします。

4.

クラスターカスタムリソースのステータスをチェックして、受付 Webhook の現在の状態を確認します。

a.

ClusterResourceOverride Operator ページで、cluster をクリックします。

b.

ClusterResourceOverride Details ページで、YAML をクリックします。Webhook の呼び出し時に、mutatingWebhookConfigurationRef セクションが表示されます。

```

apiVersion: operator.autoscaling.openshift.io/v1
kind: ClusterResourceOverride
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |

{"apiVersion":"operator.autoscaling.openshift.io/v1","kind":"ClusterResourceOverride","metadata":{"annotations":{},"name":"cluster"},"spec":{"podResourceOverride":{"spec":{"cpuRequestToLimitPercent":25,"limitCPUMemoryPercent":200,"memoryRequestToLimitPercent":50}}}}
  creationTimestamp: "2019-12-18T22:35:02Z"
  generation: 1
  name: cluster
  resourceVersion: "127622"
  selfLink:
/apis/operator.autoscaling.openshift.io/v1/clusterresourceoverrides/cluster
  uid: 978fc959-1717-4bd1-97d0-ae00ee111e8d
spec:
  podResourceOverride:
    spec:
      cpuRequestToLimitPercent: 25

```

```

limitCPUToMemoryPercent: 200
memoryRequestToLimitPercent: 50
status:

# ...

mutatingWebhookConfigurationRef: ❶
apiVersion: admissionregistration.k8s.io/v1
kind: MutatingWebhookConfiguration
name: clusterresourceoverrides.admission.autoscaling.openshift.io
resourceVersion: "127621"
uid: 98b3b8ae-d5ce-462b-8ab5-a729ea8f38f3

# ...

```

❶

ClusterResourceOverride 受付 Webhook への参照。

8.5.2.2. CLI を使用した Cluster Resource Override Operator のインストール

Red Hat OpenShift Service on AWS の CLI を使用して Cluster Resource Override Operator をインストールし、クラスターでオーバーコミットを制御できます。

前提条件

- 制限がコンテナに設定されていない場合、Cluster Resource Override Operator は影響を与えません。LimitRange オブジェクトを使用してプロジェクトのデフォルト制限を指定するか、Pod 仕様で制限を設定して上書きが適用されるようにする必要があります。

手順

CLI を使用して Cluster Resource Override Operator をインストールするには、以下を実行します。

1. Cluster Resource Override の namespace を作成します。
 - a. Cluster Resource Override Operator の Namespace オブジェクト YAML ファイル (cro-namespace.yaml など) を作成します。

```

apiVersion: v1
kind: Namespace
metadata:
  name: clusterresourceoverride-operator

```

- b. namespace を作成します。

```
$ oc create -f <file-name>.yaml
```

以下に例を示します。

```
$ oc create -f cro-namespace.yaml
```

2. Operator グループを作成します。

- a. Cluster Resource Override Operator の OperatorGroup オブジェクトの YAML ファイル (cro-og.yaml など) を作成します。

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: clusterresourceoverride-operator
  namespace: clusterresourceoverride-operator
spec:
  targetNamespaces:
    - clusterresourceoverride-operator
```

- b. Operator グループを作成します。

```
$ oc create -f <file-name>.yaml
```

以下に例を示します。

```
$ oc create -f cro-og.yaml
```

3. サブスクリプションを作成します。

- a. Cluster Resource Override Operator の Subscription オブジェクト YAML ファイル (cro-sub.yaml など) を作成します。

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
```

```

name: clusterresourceoverride
namespace: clusterresourceoverride-operator
spec:
  channel: "4"
  name: clusterresourceoverride
  source: redhat-operators
  sourceNamespace: openshift-marketplace

```

- b. サブスクリプションを作成します。

```
$ oc create -f <file-name>.yaml
```

以下に例を示します。

```
$ oc create -f cro-sub.yaml
```

4. ClusterResourceOverride カスタムリソース (CR) オブジェクトを clusterresourceoverride-operator namespace に作成します。

- a. clusterresourceoverride-operator namespace に切り替えます。

```
$ oc project clusterresourceoverride-operator
```

- b. Cluster Resource Override Operator の ClusterResourceOverride オブジェクト YAML ファイル (cro-cr.yaml など) を作成します。

```

apiVersion: operator.autoscaling.openshift.io/v1
kind: ClusterResourceOverride
metadata:
  name: cluster 1
spec:
  podResourceOverride:
    spec:
      memoryRequestToLimitPercent: 50 2
      cpuRequestToLimitPercent: 25 3
      limitCPUToMemoryPercent: 200 4

```

1

名前は cluster でなければなりません。

2

3

オプション: コンテナ CPU の制限を上書きするためのパーセンテージが使用される場合は、これを 1-100 までの値で指定します。デフォルトは 25 です。

4

オプション: コンテナメモリの制限を上書きするためのパーセンテージが使用される場合は、これを指定します。1Gi の RAM の 100 パーセントでのスケールアップは、1 CPU コアに等しくなります。これは、CPU 要求を上書きする前に処理されます (設定されている場合)。デフォルトは 200 です。

c.

ClusterResourceOverride オブジェクトを作成します。

```
$ oc create -f <file-name>.yaml
```

以下に例を示します。

```
$ oc create -f cro-cr.yaml
```

5.

クラスターカスタムリソースのステータスをチェックして、受付 Webhook の現在の状態を確認します。

```
$ oc get clusterresourceoverride cluster -n clusterresourceoverride-operator -o yaml
```

Webhook の呼び出し時に、mutatingWebhookConfigurationRef セクションが表示されます。

出力例

```
apiVersion: operator.autoscaling.openshift.io/v1
kind: ClusterResourceOverride
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"operator.autoscaling.openshift.io/v1","kind":"ClusterResourceOverride",
      "metadata":{"annotations":{},"name":"cluster"},"spec":{"podResourceOverride":
      {"spec":
      {"cpuRequestToLimitPercent":25,"limitCPUToMemoryPercent":200,"memoryRequestToLimitPercent":50}}}}
```

```
creationTimestamp: "2019-12-18T22:35:02Z"
generation: 1
name: cluster
resourceVersion: "127622"
selfLink: /apis/operator.autoscaling.openshift.io/v1/clusterresourceoverrides/cluster
uid: 978fc959-1717-4bd1-97d0-ae00ee111e8d
spec:
  podResourceOverride:
    spec:
      cpuRequestToLimitPercent: 25
      limitCPUMemoryPercent: 200
      memoryRequestToLimitPercent: 50
status:

# ...

mutatingWebhookConfigurationRef: 1
  apiVersion: admissionregistration.k8s.io/v1
  kind: MutatingWebhookConfiguration
  name: clusterresourceoverrides.admission.autoscaling.openshift.io
  resourceVersion: "127621"
  uid: 98b3b8ae-d5ce-462b-8ab5-a729ea8f38f3

# ...
```

1

ClusterResourceOverride 受付 Webhook への参照。

8.5.2.3. クラスターレベルのオーバーコミットの設定

Cluster Resource Override Operator には、Operator がオーバーコミットを制御する必要のある各プロジェクトの ClusterResourceOverride カスタムリソース (CR) およびラベルが必要です。

前提条件

- 制限がコンテナに設定されていない場合、Cluster Resource Override Operator は影響を与えません。LimitRange オブジェクトを使用してプロジェクトのデフォルト制限を指定するか、Pod 仕様で制限を設定して上書きが適用されるようにする必要があります。

手順

クラスターレベルのオーバーコミットを変更するには、以下を実行します。

1.

ClusterResourceOverride CR を編集します。

```

apiVersion: operator.autoscaling.openshift.io/v1
kind: ClusterResourceOverride
metadata:
  name: cluster
spec:
  podResourceOverride:
    spec:
      memoryRequestToLimitPercent: 50 ①
      cpuRequestToLimitPercent: 25 ②
      limitCPUToMemoryPercent: 200 ③
# ...

```

①

オプション: コンテナメモリの制限を上書きするためのパーセンテージが使用される場合は、これを 1-100 までの値で指定します。デフォルトは 50 です。

②

オプション: コンテナ CPU の制限を上書きするためのパーセンテージが使用される場合は、これを 1-100 までの値で指定します。デフォルトは 25 です。

③

オプション: コンテナメモリの制限を上書きするためのパーセンテージが使用される場合は、これを指定します。1Gi の RAM の 100 パーセントでのスケーリングは、1 CPU コアに等しくなります。これは、CPU 要求を上書きする前に処理されます (設定されている場合)。デフォルトは 200 です。

2.

以下のラベルが Cluster Resource Override Operator がオーバーコミットを制御する必要がある各プロジェクトの namespace オブジェクトに追加されていることを確認します。

```

apiVersion: v1
kind: Namespace
metadata:
# ...

labels:
  clusterresourceoverrides.admission.autoscaling.openshift.io/enabled: "true" ①
# ...

```

①

8.5.3. ノードレベルのオーバーコミット

QoS (Quality of Service) 保証、CPU 制限、またはリソースの予約など、特定ノードでオーバーコミットを制御するさまざまな方法を使用できます。特定のノードおよび特定のプロジェクトのオーバーコミットを無効にすることもできます。

8.5.3.1. コンピュートリソースとコンテナについて

コンピュートリソースについてのノードで実施される動作は、リソースタイプによって異なります。

8.5.3.1.1. コンテナの CPU 要求について

コンテナには要求する CPU の量が保証され、さらにコンテナで指定される任意の制限までノードで利用可能な CPU を消費できます。複数のコンテナが追加の CPU の使用を試行する場合、CPU 時間が各コンテナで要求される CPU の量に基づいて分配されます。

たとえば、あるコンテナが 500m の CPU 時間を要求し、別のコンテナが 250m の CPU 時間を要求した場合、ノードで利用可能な追加の CPU 時間は 2:1 の比率でコンテナ間で分配されます。コンテナが制限を指定している場合、指定した制限を超えて CPU を使用しないようにスロットリングされます。CPU 要求は、Linux カーネルの CFS 共有サポートを使用して適用されます。デフォルトで、CPU 制限は、Linux カーネルの CFS クォータサポートを使用して 100ms の測定間隔で適用されます。ただし、これは無効にすることができます。

8.5.3.1.2. コンテナのメモリー要求について

コンテナには要求するメモリー量が保証されます。コンテナは要求したよりも多くのメモリーを使用できますが、いったん要求した量を超えた場合には、ノードのメモリーが不足している状態では強制終了される可能性があります。コンテナが要求した量よりも少ないメモリーを使用する場合、システムタスクやデーモンがノードのリソース予約で確保されている分よりも多くのメモリーを必要としない限りそれが強制終了されることはありません。コンテナがメモリーの制限を指定する場合、その制限量を超えると即時に強制終了されます。

8.5.3.2. オーバーコミットメントと QoS (Quality of Service) クラスについて

ノードは、要求を指定しない Pod がスケジュールされている場合やノードのすべての Pod での制限の合計が利用可能なマシンの容量を超える場合に オーバーコミット されます。

オーバーコミットされる環境では、ノード上の Pod がいずれかの時点で利用可能なコンピュートリ

ソースよりも多くの量の使用を試行することができます。これが生じると、ノードはそれぞれの Pod に優先順位を指定する必要があります。この決定を行うために使用される機能は、QoS (Quality of Service) クラスと呼ばれます。

Pod は、優先度の高い順に 3 つの QoS クラスの 1 つとして指定されます。

表8.19 QoS (Quality of Service) クラス

| 優先順位 | クラス名 | 説明 |
|--------|------------|--|
| 1 (最高) | Guaranteed | 制限およびオプションの要求がすべてのリソースについて設定されている場合 (0 と等しくない) でそれらの値が等しい場合、Pod は Guaranteed として分類されます。 |
| 2 | Burstable | 制限およびオプションの要求がすべてのリソースについて設定されている場合 (0 と等しくない) でそれらの値が等しくない場合、Pod は Burstable として分類されます。 |
| 3 (最低) | BestEffort | 要求および制限がリソースのいずれについても設定されない場合、Pod は BestEffort として分類されます。 |

メモリーは圧縮できないリソースであるため、メモリー不足の状態では、最も優先順位の低いコンテナが最初に強制終了されます。

- Guaranteed コンテナは優先順位が最も高いコンテナとして見なされ、保証されます。強制終了されるのは、これらのコンテナで制限を超えるか、システムがメモリー不足の状態にあるものの、エビクトできる優先順位の低いコンテナが他にない場合のみです。
- システム不足の状態にある Burstable コンテナは、制限を超過し、BestEffort コンテナが他に存在しない場合に強制終了される可能性があります。
- BestEffort コンテナは優先順位の最も低いコンテナとして処理されます。これらのコンテナのプロセスは、システムがメモリー不足になると最初に強制終了されます。

8.5.3.2.1. Quality of Service (QoS) 層でのメモリーの予約方法について

qos-reserved パラメーターを使用して、特定の QoS レベルの Pod で予約されるメモリーのパーセンテージを指定することができます。この機能は、最も低い QoS クラスの Pod が高い QoS クラスの Pod で要求されるリソースを使用できないようにするために要求されたリソースの予約を試行します。

Red Hat OpenShift Service on AWS は、`qos-reserved` パラメーターを次のように使用します。

- `qos-reserved=memory=100%` の値は、**Burstable** および **BestEffort QoS** クラスが、これらより高い QoS クラスで要求されたメモリーを消費するのを防ぎます。これにより、**Guaranteed** および **Burstable** ワークロードのメモリーリソースの保証レベルを上げることが優先され、**BestEffort** および **Burstable** ワークロードでの OOM が発生するリスクが高まります。
- `qos-reserved=memory=50%` の値は、**Burstable** および **BestEffort QoS** クラスがこれらより高い QoS クラスによって要求されるメモリーの半分を消費することを許可します。
- `qos-reserved=memory=0%` の値は、**Burstable** および **BestEffort QoS** クラスがノードの割り当て可能分を完全に消費することを許可しますが (利用可能な場合)、これにより、**Guaranteed** ワークロードが要求したメモリーにアクセスできなくなるリスクが高まります。この状況により、この機能は無効にされています。

8.5.3.3. swap メモリーと QOS について

QoS (Quality of Service) 保証を維持するため、`swap` はノード上でデフォルトで無効にすることができます。そうしない場合、ノードの物理リソースがオーバーサブスクライブし、Pod の配置時の Kubernetes スケジューラーによるリソース保証が影響を受ける可能性があります。

たとえば、2つの **Guaranteed pod** がメモリー制限に達した場合、それぞれのコンテナが `swap` メモリーを使用し始める可能性があります。十分な `swap` 領域がない場合には、`pod` のプロセスはシステムのオーバーサブスクライブのために終了する可能性があります。

`swap` を無効にしないと、ノードが **MemoryPressure** にあることを認識しなくなり、Pod がスケジューリング要求に対応するメモリーを受け取れなくなります。結果として、追加の Pod がノードに配置され、メモリー不足の状態が加速し、最終的にはシステムの **Out Of Memory (OOM)** イベントが発生するリスクが高まります。



重要

`swap` が有効にされている場合、利用可能なメモリーについてのリソース不足の処理 (**out of resource handling**) のエビクシオンしきい値は予期どおりに機能しなくなります。メモリー不足の状態の場合に Pod をノードからエビクトし、Pod を不足状態にない別のノードで再スケジューリングできるようにリソース不足の処理 (**out of resource handling**) を利用できるようにします。

8.5.3.4. ノードのオーバーコミットについて

オーバーコミット環境では、最適なシステム動作を提供できるようにノードを適切に設定する必要があります。

ノードが起動すると、メモリー管理用のカーネルの調整可能なフラグが適切に設定されます。カーネルは、物理メモリーが不足しない限り、メモリーの割り当てに失敗することはありません。

この動作を確実にするために、Red Hat OpenShift Service on AWS は、`vm.overcommit_memory` パラメーターを 1 に設定し、デフォルトのオペレーティングシステム設定をオーバーライドすることで、常にメモリーをオーバーコミットするようにカーネルを設定します。

また、Red Hat OpenShift Service on AWS は、`vm.panic_on_oom` パラメーターを 0 に設定することで、メモリー不足時のカーネルパニックを回避します。0 の設定は、Out of Memory (OOM) 状態のときに `oom_killer` を呼び出すようカーネルに指示します。これにより、優先順位に基づいてプロセスを強制終了します。

現在の設定は、ノードに以下のコマンドを実行して表示できます。

```
$ sysctl -a |grep commit
```

出力例

```
#...  
vm.overcommit_memory = 0  
#...
```

```
$ sysctl -a |grep panic
```

出力例

```
#...  
vm.panic_on_oom = 0  
#...
```



注記

上記のフラグはノード上にすでに設定されているはずであるため、追加のアクションは不要です。

各ノードに対して以下の設定を実行することもできます。

- CPU CFS クォータを使用した CPU 制限の無効化または実行
- システムプロセスのリソース予約
- Quality of Service (QoS) 層でのメモリー予約

8.5.3.5. CPU CFS クォータの使用による CPU 制限の無効化または実行

デフォルトで、ノードは Linux カーネルの Completely Fair Scheduler (CFS) クォータのサポートを使用して、指定された CPU 制限を実行します。

CPU 制限の適用を無効にする場合、それがノードに与える影響を理解しておくことが重要になります。

- コンテナに CPU 要求がある場合、これは Linux カーネルの CFS 共有によって引き続き適用されます。
- コンテナに CPU 要求がなく、CPU 制限がある場合は、CPU 要求はデフォルトで指定される CPU 制限に設定され、Linux カーネルの CFS 共有によって適用されます。
- コンテナに CPU 要求と制限の両方がある場合、CPU 要求は Linux カーネルの CFS 共有によって適用され、CPU 制限はノードに影響を与えません。

前提条件

- 次のコマンドを入力して、設定するノードタイプの静的な MachineConfigPool CRD に関連付けられたラベルを取得します。

```
$ oc edit machineconfigpool <name>
```

以下に例を示します。

```
$ oc edit machineconfigpool worker
```

出力例

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  creationTimestamp: "2022-11-16T15:34:25Z"
  generation: 4
  labels:
    pools.operator.machineconfiguration.openshift.io/worker: "" 1
  name: worker
```

1

Labels の下にラベルが表示されます。

ヒント

ラベルが存在しない場合は、次のようなキー/値のペアを追加します。

```
$ oc label machineconfigpool worker custom-kubelet=small-pods
```

手順

1. 設定変更のためのカスタムリソース (CR) を作成します。

CPU 制限を無効化する設定例

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: disable-cpu-units ❶
spec:
  machineConfigPoolSelector:
    matchLabels:
      pools.operator.machineconfiguration.openshift.io/worker: "" ❷
  kubeletConfig:
    cpuCfsQuota: false ❸
```

❶

CR に名前を割り当てます。

❷

マシン設定プールからラベルを指定します。

❸

cpuCfsQuota パラメーターを false に設定します。

2.

以下のコマンドを実行して CR を作成します。

```
$ oc create -f <file_name>.yaml
```

8.5.3.6. システムリソースのリソース予約

より信頼できるスケジューリングを実現し、ノードリソースのオーバーコミットメントを最小化するために、各ノードでは、クラスターが機能できるようノードで実行する必要があるシステムデーモン用にそのリソースの一部を予約することができます。とくに、メモリーなどの圧縮できないリソースのリソースを予約することが推奨されます。

手順

Pod 以外のプロセスのリソースを明示的に予約するには、スケジューリングで利用可能なリソースを指定することにより、ノードリソースを割り当てます。詳細については、ノードのリソースの割り当

てを参照してください。

8.5.3.7. ノードのオーバーコミットの無効化

有効にされているオーバーコミットを、各ノードで無効にできます。

手順

ノード内のオーバーコミットを無効にするには、そのノード上で以下のコマンドを実行します。

```
$ sysctl -w vm.overcommit_memory=0
```

8.5.4. プロジェクトレベルの制限

オーバーコミットを制御するには、プロジェクトごとのリソース制限の範囲を設定し、オーバーコミットが超過できないプロジェクトのメモリーおよび CPU 制限およびデフォルト値を指定できます。

プロジェクトレベルのリソース制限の詳細は、関連情報を参照してください。

または、特定のプロジェクトのオーバーコミットを無効にすることもできます。

8.5.4.1. プロジェクトでのオーバーコミットメントの無効化

有効にされているオーバーコミットメントをプロジェクトごとに無効にすることができます。たとえば、インフラストラクチャーコンポーネントはオーバーコミットメントから独立して設定できます。

手順

プロジェクト内のオーバーコミットメントを無効にするには、以下の手順を実行します。

1. `namespace` オブジェクトファイルを編集します。
2. 以下のアノテーションを追加します。

```
apiVersion: v1  
kind: Namespace
```



```
metadata:  
  annotations:  
    quota.openshift.io/cluster-resource-override-enabled: "false" 1  
# ...
```

1

このアノテーションを `false` に設定すると、この namespace のオーバーコミットが無効になります。