OpenShift Dedicated 4

ノード

OpenShift Dedicated 4 でのノードの設定および管理

Last Updated: 2020-05-15

OpenShift Dedicated 4 ノード

OpenShift Dedicated 4 でのノードの設定および管理

法律上の通知

概要

本書では、クラスターのノード、Pod、コンテナーを使用する方法について説明します。

目次

第1章 POD の使用

1.1. POD の使用

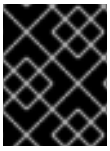
Pod は 1 つのホストにデプロイされる 1 つ以上のコンテナーであり、定義され、デプロイされ、管理される最小のコンピュート単位です。

1.1.1. Pod について

Pod はコンテナー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 には独自の内部 IP アドレスが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 内のコンテナー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、コンテナーが終了するまで実行されるか、その他の理由でコンテナー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、コンテナー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。

OpenShift Dedicated は Pod をほとんどがイミュータブルなものとして処理します。Pod が実行中の場合は Pod に変更を加えることができません。OpenShift Dedicated は既存 Pod を終了し、これを変更された設定、ベースイメージのいずれかまたはその両方で再作成して変更を実装します。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はく、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。



重要

OpenShift Dedicated ノードホストごとの推奨される Pod の最大数は 35 です。1 つのノードに 40 を超える Pod を加えることはできません。

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。

1.1.2. Pod 設定の例

OpenShift Dedicated は、**Pod** の Kubernetes の概念を活用しています。これはホスト上に共にデプロイされる 1 つ以上のコンテナーであり、定義され、デプロイされ、管理される最小のコンピュート単位です。

以下は Pod のサンプル定義です。これは、統合コンテナーイメージレジストリーという OpenShift Dedicated インフラストラクチャーの一部で、長期間実行されるサービスを提供します。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
  namespace: default
  selfLink: /api/v1/namespaces/default/pods/example
  uid: 5cc30063-0265780783bc
  resourceVersion: '165032'
  creationTimestamp: '2019-02-13T20:31:37Z'
  labels:                                    (1)
    app: hello-openshift
  annotations:
    openshift.io/scc: anyuid
spec:
  restartPolicy: Always                      (2)
  serviceAccountName: default
  imagePullSecrets:
    - name: default-dockercfg-5zrhb
  priority: 0
  schedulerName: default-scheduler
  terminationGracePeriodSeconds: 30
  nodeName: ip-10-0-140-16.us-east-2.compute.internal
  securityContext:                           (3)
    seLinuxOptions:
      level: 's0:c11,c10'
  containers:                                (4)
    - resources: {}
      terminationMessagePath: /dev/termination-log
      name: hello-openshift
      securityContext:
        capabilities:
          drop:
            - MKNOD
        procMount: Default
      ports:
        - containerPort: 8080
          protocol: TCP
      imagePullPolicy: Always
      volumeMounts:                          (5)
        - name: default-token-wbqsl
          readOnly: true
          mountPath: /var/run/secrets/kubernetes.io/serviceaccount
      terminationMessagePolicy: File
      image: registry.redhat.io/openshift4/ose-ogging-eventrouter:v4.3 (6)
  serviceAccount: default                    (7)
  volumes:                                   (8)
    - name: default-token-wbqsl
      secret:
        secretName: default-token-wbqsl
        defaultMode: 420
  dnsPolicy: ClusterFirst
status:
  phase: Pending
  conditions:
    - type: Initialized
      status: 'True'
      lastProbeTime: null
```

```
    lastTransitionTime: '2019-02-13T20:31:37Z'
  - type: Ready
    status: 'False'
    lastProbeTime: null
    lastTransitionTime: '2019-02-13T20:31:37Z'
    reason: ContainersNotReady
    message: 'containers with unready status: [hello-openshift]'
  - type: ContainersReady
    status: 'False'
    lastProbeTime: null
    lastTransitionTime: '2019-02-13T20:31:37Z'
    reason: ContainersNotReady
    message: 'containers with unready status: [hello-openshift]'
  - type: PodScheduled
    status: 'True'
    lastProbeTime: null
    lastTransitionTime: '2019-02-13T20:31:37Z'
hostIP: 10.0.140.16
startTime: '2019-02-13T20:31:37Z'
containerStatuses:
  - name: hello-openshift
    state:
      waiting:
        reason: ContainerCreating
    lastState: {}
    ready: false
    restartCount: 0
    image: openshift/hello-openshift
    imageID: ''
qosClass: BestEffort
```

1. 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。この例で使用されているラベルは **registry=default** です。

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. OpenShift Dedicated は、コンテナーが特権付きコンテナーとして実行されるか、選択したユーザーとして実行され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** はコンテナー定義の配列を指定します。 この場合 (ほとんどの場合)、これは 1 つのみになります。

5. コンテナーは外部ストレージボリュームがコンテナー内にマウントされるかどうかを指定します。この場合、レジストリーのデータを保存するためのボリュームと、OpenShift Dedicated API に対して要求を行うためにレジストリーが必要とする認証情報へのアクセス用のボリュームがあります。

6. Pod 内の各コンテナーは、独自のコンテナーイメージからインスタンス化されます。

7. OpenShift Dedicated API に対して要求する Pod は一般的なパターンです。この場合、**serviceAccount** フィールドがあり、これは要求を行う際に Pod が認証する必要のあるサービスアカウントユーザーを指定するために使用されます。これにより、カスタムインフラストラクチャーコンポーネントの詳細なアクセス制御が可能になります。

8 Pod は、コンテナーで使用できるストレージボリュームを定義します。この場合、レジストリーストレージの一時的なボリュームおよびサービスアカウントの認証情報が含まれる **secret** ボ

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された後に OpenShift Dedicated 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。

1.2. 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。

1.2.1. Pod について

OpenShift Dedicated は、**Pod** の Kubernetes の概念を活用しています。これはホスト上に共にデプロイされる 1 つ以上のコンテナーであり、定義され、デプロイされ、管理される最小のコンピュート単位です。Pod はコンテナー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 の一覧を表示したり、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。

1.2.2. 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 の一覧を表示できます。

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. 次のコマンドを実行します。

```
$ oc get pods
```

例:

```
$ oc get pods -n openshift-console
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

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>
```

1.2.3. Pod の使用状況についての統計の表示

コンテナー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** パーミッションがなければなりません。
- 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. 次のコマンドを実行します。

```
$ oc adm top pods
```

例:

```
$ oc adm top pods -n openshift-console
NAME                          CPU(cores)   MEMORY(bytes)
console-7f58c69899-q8c8k      0m           22Mi
console-7f58c69899-xhbgg      0m           25Mi
downloads-594fcccf94-bcxk8    3m           18Mi
downloads-594fcccf94-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。

1.3. POD の自動スケーリング

開発者として、Horizontal Pod Autoscaler (HPA) を使って、レプリケーションコントローラーに属する Pod から収集されるメトリクスまたはデプロイメント設定に基づき、OpenShift Dedicated がレプリケーションコントローラーまたはデプロイメント設定のスケールを自動的に増減する方法を指定できます。

1.3.1. Horizontal Pod Autoscaler について

Horizontal Pod Autoscaler を作成することで、実行する Pod の最小数と最大数を指定するだけでなく、Pod がターゲットに設定する CPU の使用率またはメモリー使用率を指定することができます。

重要

メモリー使用率の自動スケーリングはテクノロジープレビュー機能のみとして提供されています。

Horizontal Pod Autoscaler を作成すると、OpenShift Dedicated は Pod で CPU またはメモリーリソースのメトリクスのクエリーを開始します。メトリクスが利用可能になると、Horizontal Pod Autoscaler は必要なメトリクスの使用率に対する現在のメトリクスの使用率の割合を計算し、随時スケールアップまたはスケールダウンを実行します。クエリーとスケーリングは一定間隔で実行されますが、メトリクスが利用可能になるでに 1 分から 2 分の時間がかかる場合があります。

レプリケーションコントローラーの場合、このスケーリングはレプリケーションコントローラーのレプリカに直接対応します。デプロイメント設定の場合、スケーリングはデプロイメント設定のレプリカ数に直接対応します。自動スケーリングは **Complete** フェーズの最新デプロイメントにのみ適用されることに注意してください。

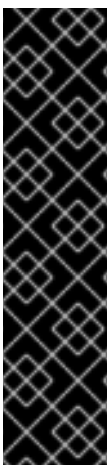
OpenShift Dedicated はリソースに自動的に対応し、起動時などのリソースの使用が急増した場合など必要のない自動スケーリングを防ぎます。**unready** 状態の Pod には、スケールアップ時の使用率が **0 CPU** と指定され、Autoscaler はスケールダウン時にはこれらの Pod を無視します。既知のメトリクスのない Pod にはスケールアップ時の使用率が **0% CPU**、スケールダウン時に **100% CPU** となります。これにより、HPA の決定時に安定性が増します。この機能を使用するには、readiness チェックを設定して新規 Pod が使用可能であるかどうかを判別します。

1.3.1.1. サポートされるメトリクス

以下のメトリクスは Horizontal Pod Autoscaler でサポートされています。

表1.1 メトリクス

メトリクス	説明	API バージョン
CPU の使用率	使用されている CPU コアの数。Pod の要求される CPU の割合の計算に使用されます。	**autoscaling/v1**, **autoscaling/v2beta2**
メモリーの使用率	使用されているメモリーの量。Pod の要求されるメモリーの割合の計算に使用されます。	**autoscaling/v2beta2**



重要

メモリーベースの自動スケーリングでは、メモリー使用量がレプリカ数と比例して増減する必要があります。平均的には以下のようになります。

- レプリカ数が増えると、Pod ごとのメモリー (作業セット) の使用量が全体的に減少します。
- レプリカ数が減ると、Pod ごとのメモリー使用量が全体的に増加します。

OpenShift Dedicated Web コンソールを使用して、アプリケーションのメモリー動作を確認し、メモリーベースの自動スケーリングを使用する前にアプリケーションがそれらの要件を満たしていることを確認します。

1.3.2. CPU 使用率のための Horizontal Pod Autoscaler の作成

指定する CPU 使用率を維持するために、オブジェクトに関連付けられた Pod を自動的にスケーリングする既存の DeploymentConfig または ReplicationController オブジェクトについて Horizontal Pod Autoscaler (HPA) を作成できます。

HPA 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。

CPU 使用率について自動スケーリングを行う際に、**oc autoscale** コマンドを使用し、実行する必要のある Pod の最小数および最大数と Pod がターゲットとして設定する必要のある平均 CPU 使用率を指定することができます。最小値を指定しない場合、Pod には OpenShift Dedicated サーバーからのデフォルト値が付与されます。特定の CPU 値について自動スケーリングを行うには、ターゲット CPU および Pod の制限のある **HorizontalPodAutoscaler** オブジェクトを作成します。

前提条件

Horizontal Pod Autoscaler を使用するには、クラスターの管理者はクラスターメトリクスを適切に設定している必要があります。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
```

```
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>
```

手順

CPU 使用率のための Horizontal Pod Autoscaler を作成するには、以下を実行します。

1. 以下のいずれかの手順を実行します。

   - CPU 使用率のパーセントに基づいてスケーリングするには、既存の DeploymentConfig の **HorizontalPodAutoscaler** オブジェクトを作成します。

     ```
     $ oc autoscale dc/<dc-name> \ (1)
     ```

```
--min <number> \ 2
--max <number> \ 3
--cpu-percent=<percent> 4
```

1. DeploymentConfig の名前を指定します。オブジェクトが存在する必要があります。
2. オプションで、スケールダウン時のレプリカの最小数を指定します。
3. スケールアップ時のレプリカの最大数を指定します。
4. 要求された CPU のパーセントで表示された、すべての Pod に対する目標の平均 CPU 使用率を指定します。指定しない場合または負の値の場合、デフォルトの自動スケーリングポリシーが使用されます。

- CPU 使用率のパーセントに基づいてスケーリングするには、既存の ReplicationController の **HorizontalPodAutoscaler** オブジェクトを作成します。

```
$ oc autoscale rc/<rc-name> 1
  --min <number> \ 2
  --max <number> \ 3
  --cpu-percent=<percent> 4
```

1. ReplicationController の名前を指定します。オブジェクトが存在する必要があります。
2. スケールダウン時のレプリカの最小数を指定します。
3. スケールアップ時のレプリカの最大数を指定します。
4. 要求された CPU のパーセントで表示された、すべての Pod に対する目標の平均 CPU 使用率を指定します。指定しない場合または負の値の場合、デフォルトの自動スケーリングポリシーが使用されます。

- 特定の CPU 値についてスケーリングするには、既存の DeploymentConfig または ReplicationController について以下のような YAML 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: autoscaling/v2beta2 1
kind: HorizontalPodAutoscaler
metadata:
  name: cpu-autoscale 2
  namespace: default
spec:
  scaleTargetRef:
    apiVersion: v1 3
    kind: ReplicationController 4
    name: example 5
  minReplicas: 1 6
  maxReplicas: 10 7
  metrics: 8
  - type: Resource
```

```
    resource:
      name: cpu 9
      target:
        type: Utilization 10
        averageValue: 500Mi 11
```

1. Use the **autoscaling/v2beta2** API.
2. この Horizontal Pod Autoscaler オブジェクトの名前を指定します。
3. スケーリングするオブジェクトの API バージョンを指定します。
   - ReplicationController については、**v1** を使用します。
   - DeploymentConfig については、**apps.openshift.io/v1** を使用します。
4. スケーリングするオブジェクトの種類 (**ReplicationController** または **DeploymentConfig** のいずれか) を指定します。
5. スケーリングするオブジェクトの名前を指定します。オブジェクトが存在する必要があります。
6. スケールダウン時のレプリカの最小数を指定します。
7. スケールアップ時のレプリカの最大数を指定します。
8. メモリー使用率に **metrics** パラメーターを使用します。
9. CPU 使用率に **cpu** を指定します。
10. **Utilization** に設定します。
11. タイプを **averageValue** に設定します。

b. Horizontal Pod Autoscaler を作成します。

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

2. Horizontal Pod Autoscaler が作成されていることを確認します。

```
$ oc get hpa hpa-resource-metrics-memory

NAME                              REFERENCE                  TARGETS        MINPODS
MAXPODS   REPLICAS   AGE
oc get hpa hpa-resource-metrics-memory   ReplicationController/example   2441216/500Mi
1         10         1          20m
```

以下のコマンドは、**image-registry** DeploymentConfig が制御する Pod の 3 から 7 までのレプリカを維持し、すべての Pod で 75% の平均 CPU 使用率を維持する Horizontal Pod Autoscaler を作成します。

```
$ oc autoscale dc/image-registry --min 3 --max 7 --cpu-percent=75
deploymentconfig "image-registry" autoscaled
```

このコマンドは、以下の定義で Horizontal Pod Autoscaler を作成します。

```
$ oc edit hpa frontend -n openshift-image-registry
```

```
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  creationTimestamp: "2020-02-21T20:19:28Z"
  name: image-registry
  namespace: default
  resourceVersion: "32452"
  selfLink: /apis/autoscaling/v1/namespaces/default/horizontalpodautoscalers/frontend
  uid: 1a934a22-925d-431e-813a-d00461ad7521
spec:
  maxReplicas: 7
  minReplicas: 3
  scaleTargetRef:
    apiVersion: apps.openshift.io/v1
    kind: DeploymentConfig
    name: image-registry
  targetCPUUtilizationPercentage: 75
status:
  currentReplicas: 5
  desiredReplicas: 0
```

以下の例は、**image-registry** DeploymentConfig の自動スケーリングを示しています。最初のデプロイメントでは 3 つの Pod が必要です。HPA オブジェクトは最小で 5 まで増加され、Pod 上の CPU 使用率が 75% に達すると、Pod を最大 7 まで増やします。

```
$ oc get dc image-registry
NAME             REVISION   DESIRED   CURRENT   TRIGGERED BY
image-registry   1          3         3         config

$ oc autoscale dc/image-registry --min=5 --max=7 --cpu-percent=75
horizontalpodautoscaler.autoscaling/image-registry autoscaled

$ oc get dc image-registry
NAME             REVISION   DESIRED   CURRENT   TRIGGERED BY
image-registry   1          5         5         config
```

1.3.3. メモリー使用率のための Horizontal Pod Autoscaler オブジェクトの作成

直接の値または要求されるメモリーのパーセンテージのいずれかで指定する平均のメモリー使用率を維持するために、オブジェクトに関連付けられた Pod を自動的にスケーリングする既存の DeploymentConfig または ReplicationController オブジェクトの Horizontal Pod Autoscaler (HPA) を作成できます。

HPA 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のレプリカ数を増減します。

メモリー使用率については、Pod の最小数および最大数と、Pod がターゲットとする平均のメモリー使用率を指定することができます。最小値を指定しない場合、Pod には OpenShift Dedicated サーバーからのデフォルト値が付与されます。

重要

メモリー使用率の自動スケーリングはテクノロジープレビュー機能としてのみ提供されています。

前提条件

Horizontal Pod Autoscaler を使用するには、クラスターの管理者はクラスターメトリクスを適切に設定している必要があります。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-10-0-129-223.compute.internal -n openshift-kube-scheduler
```

```
Name:         openshift-kube-scheduler-ip-10-0-129-223.compute.internal
Namespace:    openshift-kube-scheduler
Labels:       <none>
Annotations:  <none>
API Version:  metrics.k8s.io/v1beta1
Containers:
  Name:  scheduler
  Usage:
    Cpu:     2m
    Memory:  41056Ki
  Name:      wait-for-host-port
  Usage:
    Memory:  0
Kind:        PodMetrics
Metadata:
  Creation Timestamp:  2020-02-14T22:21:14Z
  Self Link:           /apis/metrics.k8s.io/v1beta1/namespaces/openshift-kube-scheduler/pods/openshift-kube-scheduler-ip-10-0-129-223.compute.internal
Timestamp:             2020-02-14T22:21:14Z
Window:                5m0s
Events:                <none>
```

手順

メモリー使用率の Horizontal Pod Autoscaler を作成するには、以下を実行します。

1. 以下のいずれか 1 つを含む YAML ファイルを作成します。

   - 特定のメモリー値についてスケーリングするには、既存の DeploymentConfig または ReplicationController について以下のような **HorizontalPodAutoscaler** オブジェクトを作成します。

     ```
     apiVersion: autoscaling/v2beta2 (1)
     kind: HorizontalPodAutoscaler
     metadata:
       name: hpa-resource-metrics-memory (2)
       namespace: default
     spec:
       scaleTargetRef:
         apiVersion: v1 (3)
     ```

```
    kind: ReplicationController (4)
    name: example (5)
  minReplicas: 1 (6)
  maxReplicas: 10 (7)
  metrics: (8)
  - type: Resource
    resource:
      name: memory (9)
      target:
        type: Utilization (10)
        averageValue: 500Mi (11)
```

1. Use the **autoscaling/v2beta2** API.
2. この Horizontal Pod Autoscaler オブジェクトの名前を指定します。
3. スケーリングするオブジェクトの API バージョンを指定します。
   - ReplicationController については、**v1** を使用します。
   - DeploymentConfig については、**apps.openshift.io/v1** を使用します。
4. スケーリングするオブジェクトの種類 (**ReplicationController** または **DeploymentConfig** のいずれか) を指定します。
5. スケーリングするオブジェクトの名前を指定します。オブジェクトが存在する必要があります。
6. スケールダウン時のレプリカの最小数を指定します。
7. スケールアップ時のレプリカの最大数を指定します。
8. メモリー使用率に **metrics** パラメーターを使用します。
9. メモリー使用率の **memory** を指定します。
10. タイプを **Utilization** に設定します。
11. **averageValue** および特定のメモリー値を指定します。

- パーセンテージについてスケーリングするには、以下のように **HorizontalPodAutoscaler** オブジェクトを作成します。

```
apiVersion: autoscaling/v2beta2 (1)
kind: HorizontalPodAutoscaler
metadata:
  name: memory-autoscale (2)
  namespace: default
spec:
  scaleTargetRef:
    apiVersion: apps.openshift.io/v1 (3)
    kind: DeploymentConfig (4)
    name: example (5)
  minReplicas: 1 (6)
```

```
maxReplicas: 10 7
metrics:
metrics: 8
- type: Resource
  resource:
    name: memory 9
    target:
      type: Utilization 10
      averageValue: 50 11
```

1. Use the **autoscaling/v2beta2** API.
2. この Horizontal Pod Autoscaler オブジェクトの名前を指定します。
3. スケーリングするオブジェクトの API バージョンを指定します。
   - ReplicationController については、**v1** を使用します。
   - DeploymentConfig については、**apps.openshift.io/v1** を使用します。
4. スケーリングするオブジェクトの種類 (**ReplicationController** または **DeploymentConfig** のいずれか) を指定します。
5. スケーリングするオブジェクトの名前を指定します。オブジェクトが存在する必要があります。
6. スケールダウン時のレプリカの最小数を指定します。
7. スケールアップ時のレプリカの最大数を指定します。
8. メモリー使用率に **metrics** パラメーターを使用します。
9. メモリー使用率の **memory** を指定します。
10. **Utilization** に設定します。
11. **averageUtilization** または **averageValue** およびメモリー値を指定します。

2. Horizontal Pod Autoscaler を作成します。

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

例:

```
$ oc create -f hpa.yaml

horizontalpodautoscaler.autoscaling/hpa-resource-metrics-memory created
```

3. Horizontal Pod Autoscaler が作成されていることを確認します。

```
$ oc get hpa hpa-resource-metrics-memory

NAME                          REFERENCE                TARGETS       MINPODS
```

```
MAXPODS   REPLICAS   AGE
oc get hpa hpa-resource-metrics-memory   ReplicationController/example   2441216/500Mi
1         10         1          20m
```

```
$ oc describe hpa hpa-resource-metrics-memory
Name:                       hpa-resource-metrics-memory
Namespace:                  default
Labels:                     <none>
Annotations:                <none>
CreationTimestamp:          Wed, 04 Mar 2020 16:31:37 +0530
Reference:                  ReplicationController/example
Metrics:                    ( current / target )
  resource memory on pods:   2441216 / 500Mi
Min replicas:               1
Max replicas:               10
ReplicationController pods:  1 current / 1 desired
Conditions:
  Type            Status  Reason              Message
  ----            ------  ------              -------
  AbleToScale     True    ReadyForNewScale    recommended size matches current size
  ScalingActive   True    ValidMetricFound    the HPA was able to successfully calculate a
replica count from memory resource
  ScalingLimited  False   DesiredWithinRange  the desired count is within the acceptable
range
Events:
  Type     Reason                Age               From                       Message
  ----     ------                ----              ----                       -------
  Normal   SuccessfulRescale     6m34s             horizontal-pod-autoscaler  New size: 1;
reason: All metrics below target
```

1.3.4. Horizontal Pod Autoscaler の状態条件について

状態条件セットを使用して、Horizontal Pod Autoscaler (HPA) がスケーリングできるかどうかや、現時点でこれがいずれかの方法で制限されているかどうかを判別できます。

HPA の状態条件は、自動スケーリング API の **v2beta1** バージョンで利用できます。

HPA は、以下の状態条件で応答します。

- **AbleToScale** 条件では、HPA がメトリクスを取得して更新できるか、またバックオフ関連の条件によりスケーリングが回避されるかどうかを指定します。
  - **True** 条件はスケーリングが許可されることを示します。
  - **False** 条件は指定される理由によりスケーリングが許可されないことを示します。
- **ScalingActive** 条件は、HPA が有効にされており (ターゲットのレプリカ数がゼロでない)、必要なメトリクスを計算できるかどうかを示します。
  - **True** 条件はメトリクスが適切に機能していることを示します。
  - **False** 条件は通常フェッチするメトリクスに関する問題を示します。
- **ScalingLimited** 条件は、必要とするスケールが Horizontal Pod Autoscaler の最大値または最小値によって制限されていたことを示します。

- **True** 条件は、スケーリングするためにレプリカの最小または最大数を引き上げるか、または引き下げる必要があることを示します。
- **False** 条件は、要求されたスケーリングが許可されることを示します。

```
$ oc describe hpa cm-test
Name:                         cm-test
Namespace:                    prom
Labels:                       <none>
Annotations:                  <none>
CreationTimestamp:            Fri, 16 Jun 2017 18:09:22 +0000
Reference:                    ReplicationController/cm-test
Metrics:                      ( current / target )
  "http_requests" on pods:    66m / 500m
Min replicas:                 1
Max replicas:                 4
ReplicationController pods:   1 current / 1 desired
Conditions: 1
  Type            Status    Reason            Message
  ----            ------    ------            -------
  AbleToScale     True      ReadyForNewScale  the last scale time was sufficiently old as to warrant a new scale
  ScalingActive   True      ValidMetricFound  the HPA was able to successfully calculate a replica count from pods metric http_request
  ScalingLimited  False     DesiredWithinRange  the desired replica count is within the acceptable range
Events:
```

1 Horizontal Pod Autoscaler の状況メッセージです。

以下は、スケーリングできない Pod の例です。

```
Conditions:
  Type        Status  Reason          Message
  ----        ------  ------          -------
  AbleToScale  False   FailedGetScale  the HPA controller was unable to get the target's current scale: no matches for kind "ReplicationController" in group "apps"
Events:
  Type     Reason          Age             From                    Message
  ----     ------          ----            ----                    -------
  Warning  FailedGetScale  6s (x3 over 36s)  horizontal-pod-autoscaler  no matches for kind "ReplicationController" in group "apps"
```

以下は、スケーリングに必要なメトリクスを取得できなかった Pod の例です。

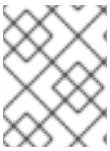
```
Conditions:
  Type               Status    Reason                 Message
  ----               ------    ------                 -------
  AbleToScale        True      SucceededGetScale      the HPA controller was able to get the target's current scale
  ScalingActive      False     FailedGetResourceMetric  the HPA was unable to compute the replica count: unable to get metrics for resource cpu: no metrics returned from heapster
```

以下は、要求される自動スケーリングが要求される最小数よりも小さい場合の Pod の例です。

```
Conditions:
  Type            Status    Reason            Message
  ----            ------    ------            -------
  AbleToScale     True      ReadyForNewScale  the last scale time was sufficiently old as to warrant a new scale
  ScalingActive   True      ValidMetricFound  the HPA was able to successfully calculate a replica count from pods metric http_request
  ScalingLimited  False     DesiredWithinRange  the desired replica count is within the acceptable range
```

1.3.4.1. Horizontal Pod Autoscaler の状態条件の表示

Pod に設定された状態条件は、Horizontal Pod Autoscaler (HPA) で表示することができます。



注記

Horizontal Pod Autoscaler の状態条件は、自動スケーリング API の **v2beta1** バージョンで利用できます。

前提条件

Horizontal Pod Autoscaler を使用するには、クラスターの管理者はクラスターメトリクスを適切に設定している必要があります。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

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>
```

手順

Pod の状態条件を表示するには、Pod の名前と共に以下のコマンドを使用します。

```
$ oc describe hpa <pod-name>
```

例:

```
$ oc describe hpa cm-test
```

条件は、出力の **Conditions** フィールドに表示されます。

```
Name:                       cm-test
Namespace:                  prom
Labels:                     <none>
Annotations:                <none>
CreationTimestamp:          Fri, 16 Jun 2017 18:09:22 +0000
Reference:                  ReplicationController/cm-test
Metrics:                    ( current / target )
  "http_requests" on pods:  66m / 500m
Min replicas:               1
Max replicas:               4
ReplicationController pods: 1 current / 1 desired
Conditions: 1
  Type            Status  Reason            Message
  ----            ------  ------            -------
  AbleToScale     True    ReadyForNewScale  the last scale time was sufficiently old as to warrant a new scale
  ScalingActive   True    ValidMetricFound  the HPA was able to successfully calculate a replica count from pods metric http_request
  ScalingLimited  False   DesiredWithinRange  the desired replica count is within the acceptable range
```

1.3.5. 追加リソース

レプリケーションコントローラーとデプロイメントコントローラーの詳細は、「Understanding Deployments and DeploymentConfigs」を参照してください。

1.4. POD への機密性の高いデータの提供

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。

1.4.1. 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、OpenShift Dedicated 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ーの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ー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。

キーのプロパティー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 シークレットオブジェクト定義

```
apiVersion: v1
kind: Secret
metadata:
  name: test-secret
  namespace: my-namespace
type: Opaque (1)
data: (2)
  username: dmFsdWUtMQ0K (3)
  password: dmFsdWUtMg0KDQo=
stringData: (4)
  hostname: myapp.mydomain.com (5)
```

(1) (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す。

(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構成されます。

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** ボリュームを使用)。

1.4.1.1. シークレットの種類

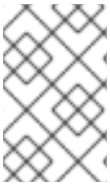
type フィールドの値で、シークレットのキー名と値の構造を指定します。このタイプを使用して、シークレットオブジェクトにユーザー名とキーの配置を実行できます。検証の必要がない場合には、デフォルト設定の **opaque** タイプを使用してください。

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はその種類のキー/値の要件に従うことが意図されていることを示します。

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。

1.4.1.2. シークレット設定の例

以下は、シークレットの設定ファイルのサンプルです。

4 つのファイルを作成する YAML シークレット

```
apiVersion: v1
kind: Secret
metadata:
  name: test-secret
data:
  username: dmFsdWUtMQ0K      (1)
  password: dmFsdWUtMQ0KDQo=  (2)
stringData:
  hostname: myapp.mydomain.com  (3)
  secret.properties: |-         (4)
    property1=valueA
    property2=valueB
```

1. デコードされる値が含まれるファイル
2. デコードされる値が含まれるファイル
3. 提供される文字列が含まれるファイル
4. 提供されるデータが含まれるファイル

シークレットデータと共にボリュ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:
  containers:
    - name: secret-test-container
```

```
      image: busybox
      command: [ "/bin/sh", "-c", "cat /etc/secret-volume/*" ]
      volumeMounts:
          # name must match the volume name below
          - name: secret-volume
            mountPath: /etc/secret-volume
            readOnly: true
  volumes:
    - name: secret-volume
      secret:
        secretName: test-secret
  restartPolicy: Never
```

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:
  containers:
    - name: secret-test-container
      image: busybox
      command: [ "/bin/sh", "-c", "export" ]
      env:
        - name: TEST_SECRET_USERNAME_ENV_VAR
          valueFrom:
            secretKeyRef:
              name: test-secret
              key: username
  restartPolicy: Never
```

シークレットデータと環境変数が投入されたビルド設定の YAML

```
apiVersion: v1
kind: BuildConfig
metadata:
  name: secret-example-bc
spec:
  strategy:
    sourceStrategy:
      env:
      - name: TEST_SECRET_USERNAME_ENV_VAR
        valueFrom:
          secretKeyRef:
            name: test-secret
            key: username
```

1.4.1.3. 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。

1.4.2. 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。

- 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** ボリュームを使用)。

1.4.2.1. 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できる必要があります。シークレットは、以下の 3 つの方法で Pod で使用されます。

- コンテナーの環境変数を事前に設定するために使用される。
- 1 つ以上のコンテナー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ーに書き込みます。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** タイプのオブジェクトを実際に参照していることを確認できる必要があります。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。

1.4.2.2. 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。 このシークレットでは、任意の値を含む、構造化されていない **key:value** ペアを利用できます。

手順

1. マスターの YAML ファイルにシークレットオブジェクトを作成します。
   例:

```
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
type: Opaque 1
```

```
data:
  username: dXNlci1uYW1l
  password: cGFzc3dvcmQ=
```

1 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シークレットオブジェクトを作成します。

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

次に、以下を実行します。

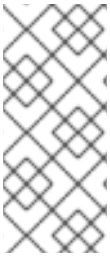
1. このシークレットを使ってこのシークレットの参照を許可したい、Pod のサービスアカウントを更新します。
2. シークレットを環境変数またはファイルとして使用する Pod を作成します (**secret** ボリュームを使用)。

1.4.3. 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、新規コンテナー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。今後はコントローラーが古い **resourceVersion** を使用して再起動できるよう 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。

1.4.4. シークレットで署名証明書を使用する方法

サービスの通信を保護するため、プロジェクト内のシークレットに追加可能な、署名されたサービス証明書/キーペアを生成するように OpenShift Dedicated を設定することができます。

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の Pod 仕様

```
apiVersion: v1
  kind: Service
  metadata:
    name: registry
```

```
    annotations:
      service.alpha.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。

1.4.4.1. 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.alpha.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.alpha.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.alpha.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。

```
$ oc get secrets

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

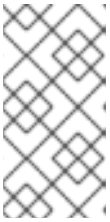
```
$ oc describe secret my-service-pod
Name:         my-service-pod
Namespace:    openshift-console
Labels:       <none>
Annotations:  kubernetes.io/service-account.name: builder
              kubernetes.io/service-account.uid: ab-11e9-988a-0eb4e1b4a396

Type:  kubernetes.io/service-account-token

Data

ca.crt:     5802 bytes
namespace:  17 bytes
token:
eyJhbGciOiJSUzI1NiIsImtpZCI6IiJ9.eyJpc3MiOiJrdWJlcm5ldGVzL3NlcnZpY2VhY2NvdW50Ii
wia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9uYW1lc3BhY2UiOiJvcGVuc2hpZnQtY29uc
29sZSIsImt1YmVyb
cnZpY2VhY2NvdW50L3NlcnZpY2UtYWNjb3VudC51aWQiOiJhYmE4Y2UyZC00MzVlLTExZTktOTg4YS0wZWI0ZTFiNGEz
OTYiLCJzdWIiOiJzeXN0ZW06c2VydmljZWFjY291bnQ6b3BlbnNoaWZ
```

5. このシークレットを使って Pod 仕様を編集します。

```
apiVersion: v1
kind: Pod
metadata:
  name: my-service-pod
spec:
  containers:
  - name: mypod
    image: redis
    volumeMounts:
    - name: foo
      mountPath: "/etc/foo"
  volumes:
  - name: foo
    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.alpha.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。

1.4.5. 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.alpha.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、またはサービスに異なる **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.alpha.openshift.io/serving-cert-generation-error**、 **service.alpha.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。

```
$ oc delete secret <secret_name>
$ oc annotate service <service_name> service.alpha.openshift.io/serving-cert-generation-error-
$ oc annotate service <service_name> service.alpha.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の後に **-** を付けます。

第2章 ジョブと DEAMONSET の使用

2.1. ジョブの使用による POD でのタスクの実行

ジョブは、OpenShift Dedicated クラスターの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 の情報を使っ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。

ジョブについての詳細は、Kubernetes のドキュメント を参照してください。

2.1.1. 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 の情報を使っ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。

OpenShift Dedicated 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プロセスを持つジョブ。
  - OpenShift Dedicated は Pod を調整し、それぞれの機能を判別するか、または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

Cron ジョブは、Cron ジョブを使って複数回実行するようにスケジュールすることが可能です。

Cron ジョブ は、ジョブの実行方法をユーザーに指定させることで、定期的なジョブにビルドされます。Cron 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。

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。

警告

Cron ジョブはスケジュールの実行時間ごとに約 1 回ずつジョブ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可能性のある場合があります。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。

2.1.2. 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。

- OpenShift Dedicated 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。
  - ワークキューを使用する並列ジョブでは、未設定のままにしておきます。未設定の場合、デフォルトは **parallelism** 値に設定されます。

2.1.2.1. 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。このフィールドは秒単位で指定し、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、最長期間は有効ではありません。

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、OpenShift Dedicated がジョブを終了します。

2.1.2.2. 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。

2.1.2.3. 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 つのフィールドは 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.successfulJobsHistoryLimit**。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)。

ヒント

- 必要がなくなった Cron ジョブを削除します。

```
$ oc delete cronjob/<cron_job_name>
```

これを実行することで、不要なアーティファクトの生成を防げます。

- **spec.suspend** を true に設定することで、その後の実行を中断することができます。その後のすべての実行は、**false** に再設定するまで中断されます。

2.1.3. 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。

2.1.4. ジョブの作成

ジョブオブジェクトを作成して OpenShift Dedicated にジョブを作成します。

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. オプションで、ジョブが並列で実行される Pod のレプリカ数を指定します。デフォルトは **1** に設定されます。
   - 非並列ジョブジョブの場合は、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. オプションで、ジョブを完了 (completed) としてマークするために必要な Pod の正常な完了数を指定します。
   - 非並列ジョブジョブの場合は、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. オプションで、ジョブを実行できる最長期間を指定します。
4. オプションで、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。

OpenShift Dedicated が失敗したコンテナーについて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。

1. ジョブを作成します。

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

注記

oc run 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 run pi --image=perl --replicas=1  --restart=OnFailure \
    --command -- perl -Mbignum=bpi -wle 'print bpi(2000)'
```

2.1.5. cron ジョブの作成

ジョブオブジェクトを作成して OpenShift Dedicated に cron ジョブを作成します。

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/v1beta1
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 1 1 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 2 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 3 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 4 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。

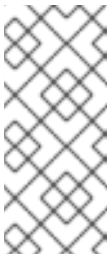
5 5 5 保持する成功した終了済みジョブの数 (デフォルトは 3 に設定)。

6 6 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 run 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 run pi --image=perl --schedule='*/1 * * * *' \
    --restart=OnFailure --labels parent="cronjobpi" \
    --command -- perl -Mbignum=bpi -wle 'print bpi(2000)'
```

oc run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。

Cron ジョブの作成時に、**oc run** は **Never** または **OnFailure** 再起動ポリシー (**--restart**) のみをサポートします。

第3章 コンテナーの使用

3.1. コンテナーについて

OpenShift Dedicated アプリケーションの基本的な単位は**コンテナー**と呼ばれています。Linux コンテナーテクノロジーは、指定されたリソースのみとの対話に制限されるように、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ーで実行される可能性があります。通常、コンテナーは任意のワークロードに使用されますが、各コンテナーは 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ーテクノロジーの各種機能を統合してきました。OpenShift Dedicated および Kubernetes は複数ホストのインストール間でコンテナーのオーケストレーションを実行する機能を追加します。

3.2. ボリュームの使用によるコンテナーデータの永続化

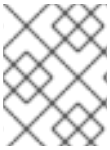
コンテナー内のファイルは一時的なものです。そのため、コンテナーがクラッシュしたり停止したりした場合は、データが失われます。**ボリューム** を使用すると、Pod 内のコンテナーが使用しているデータを永続化できます。ボリュームはディレクトリーであり、Pod 内のコンテナーからアクセスすることができます。 ここでは、データが Pod の有効期間中保存されます。

3.2.1. ボリュームについて

ボリュームとは Pod およびコンテナー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。コンテナー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。

ボリュームのファイルシステムにエラーが含まれないようにし、かつエラーが存在する場合はそれを修復するために、OpenShift Dedicated は **mount** ユーティリティーの前に **fsck** ユーティリティーを起動します。これはボリュームを追加するか、または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ーです。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。

3.2.2. OpenShift Dedicated 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 DeploymentConfig のような 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一覧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 コマンド内の **object_seletion** に、以下のいずれかを指定します。

表3.1 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>	所定のラベルセレ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 コマンドに **operation** の **--add**、**--remove**、または **--list** を指定します。

必須パラメーター

いずれの **<mandatory_parameters>** 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 **<options>** 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。

3.2.3. Pod のボリュームとボリュームマウントの一覧表示

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一覧表示することができます。

手順

ボリュームを一覧表示するには、以下の手順を実行します。

```
$ oc set volume <object_type>/<name> --list [options]
```

ボリュームのサポートされているオプションを一覧表示します。

オプション	説明	デフォルト
--name	ボリュームの名前。	
-c, --containers	名前でコンテナーを選択します。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。	**'*'**

例:

- Pod **p1** のすべてのボリュームを一覧表示するには、以下を実行します。

```
$ oc set volume pod/p1 --list
```

- すべての DeploymentConfig で定義されるボリューム **v1** を一覧表示するには、以下の手順を実行します。

```
$ oc set volume dc --all --name=v1
```

3.2.4. 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3.2 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	名前でコンテナーを選択します。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。	**'*'**

オプション	説明	デフォルト
-m, --mount-path	選択されたコンテナー内のマウントパス。	
--path	ホストパス。**--type=hostPath** の必須パラメーターです。	
--secret-name	シークレットの名前。**--type=secret** の必須パラメーターです。	
--configmap-name	configmap の名前。**--type=configmap** の必須のパラメーターです。	
--claim-name	Persistent Volume Claim (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**

例:

- 新規ボリュームソース **emptyDir** をデプロイメント設定の **レジストリー** に追加するには、以下の手順を実行します。

```
$ oc set volume dc/registry --add
```

- 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、コンテナー内の **/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
```

- 要求名 **pvc1** を使っ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ー **c1** の **/data** にマウントし、サーバー上でデプロイメント設定を更新するには、以下の手順を実行します。

  –

```
$ oc set volume -f dc.json --add --name=v1 --type=persistentVolumeClaim \
  --claim-name=pvc1 --mount-path=/data --containers=c1
```

- 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 リポジトリー **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"
            }}'
```

3.2.5. 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修正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]
```

例:

- 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 Persistent Volume Claim (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
```

- DeploymentConfig **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
```

3.2.6. 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3.3 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。	

オプション	説明	デフォルト
-c, --containers	名前でコンテナーを選択します。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**

例:

- DeploymentConfig **d1** からボリューム **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
```

- DeploymentConfig **d1** のコンテナー **c1** からボリューム **v1** をアンマウントし、**d1** のコンテナー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
```

3.2.7. Pod 内での複数の用途のためのボリュームの設定

ボリュームを設定して、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ことができます。この場合、**volumeMounts.subPath** プロパティーを使用し、ボリュームのルートの代わりにボリューム内に **subPath** を指定します。

手順

1. ボリューム内のファイルの一覧を表示して、**oc rsh** コマンドを実行します。

```
$ oc rsh <pod>
sh-4.3$ ls /path/to/volume/subpath/mount
example_file1 example_file2 example_file3
```

2. **subPath** を指定します。

subPath の使用例

```
apiVersion: v1
kind: Pod
metadata:
  name: my-site
spec:
    containers:
    - name: mysql
      image: mysql
      volumeMounts:
      - mountPath: /var/lib/mysql
        name: site-data
        subPath: mysql (1)
    - name: php
      image: php
      volumeMounts:
      - mountPath: /var/www/html
        name: site-data
        subPath: html (2)
    volumes:
    - name: site-data
      persistentVolumeClaim:
        claimName: my-site-data
```

1. データベースは **mysql** フォルダーに保存されます。
2. HTML コンテンツは **html** フォルダーに保存されます。

3.3. PROJECTED ボリュームによるボリュームのマッピング

Projected ボリューム は、いくつかの既存のボリュームソースを同じディレクトリーにマップします。

以下のタイプのボリュームソースを展開できます。

- 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。

3.3.1. Projected ボリュームについて

Projected ボリュームはこれらのボリュ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。

- 単一ボリュームを、複数のシークレットのキー、configmap、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、単一ボリュームを複数シークレットのキー、configmap、および Downward API 情報で設定し、ユーザーがボリュームの内容を完全に制御できるようにします。

以下の一般的なシナリオは、展開されるプロジェクトを使用する方法について示しています。

ConfigMap、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ー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 OpenStack を Kubernetes にデプロイする可能性があります。設定データは、サービスが実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 に実稼働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 OpenStack 設定を生成できます。

ConfigMap + 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ーをデプロイできます。たとえば、configmap 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、オペレーター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。

3.3.1.1. 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 configmap を含む Pod

```
apiVersion: v1
kind: Pod
metadata:
  name: volume-test
spec:
  containers:
  - name: container-test
    image: busybox
    volumeMounts: 1
    - name: all-in-one
      mountPath: "/projected-volume" 2
      readOnly: true 3
  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ーの **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. それぞれの Projected ボリュームソースを一覧表示するために **volumes** ブロックを追加します。
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一覧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-config** に置かれます。
9. Downward API ソースを追加します。
10. ConfigMap ソースを追加します。
11. 特定の展開におけるモードを設定します。

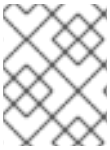
注記

Pod に複数のコンテナーがある場合、それぞれのコンテナーには **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:
  containers:
  - name: container-test
    image: busybox
```

```
  volumeMounts:
  - name: all-in-one
    mountPath: "/projected-volume"
    readOnly: true
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 は展開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展開についての **mode** を明示的に指定できます。

3.3.1.2. 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:
  containers:
  - name: container-test
    image: busybox
    volumeMounts:
    - name: all-in-one
      mountPath: "/projected-volume"
      readOnly: true
  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展開されるパスである場合の競合

自動的に展開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。

3.3.2. 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。その後に、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 つのコンテナーを実行する Pod を作成します。

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. 以下を入力し、パスワードおよびユーザー情報を適宜置き換えて、シークレットを含むファイルを作成します。

```
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
type: Opaque
data:
  pass: MWYyZDFlMmU2N2Rm
  user: YWRtaW4=
```

user および **pass** の値には、**base64** でエンコーティングされた任意の有効な文字列を使用できます。ここで使用される例は base64 でエンコーディングされた値 (**user: admin**、**pass:1f2d1e2e67df**) になります。

```
$ echo -n "admin" | base64
YWRtaW4=
$ echo -n "1f2d1e2e67df" | base64
MWYyZDFlMmU2N2Rm
```

2. 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>
```

例:

```
$ oc create -f secret.yaml
secret "mysecret" created
```

3. 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>
$ oc get secret <secret-name> -o yaml
```

例:

```
$ oc get secret mysecret
NAME       TYPE      DATA      AGE
mysecret   Opaque    2         17h
```

```
$ oc get secret mysecret -o yaml
```

```
apiVersion: v1
data:
  pass: MWYyZDFlMmU2N2Rm
  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
```

4. **volumes** セクションが含まれる以下のような Pod 設定ファイルを作成します。

```
apiVersion: v1
kind: Pod
metadata:
  name: test-projected-volume
spec:
  containers:
  - name: test-projected-volume
    image: busybox
    args:
    - sleep
    - "86400"
    volumeMounts:
    - name: all-in-one
      mountPath: "/projected-volume"
      readOnly: true
  volumes:
  - name: all-in-one
```

```
    projected:
      sources:
      - secret:      (1)
          name: user
      - secret:      (2)
          name: pass
```

(1)(2) 作成されたシークレットの名前。

5. 設定ファイルから Pod を作成します。

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

例:

```
$ oc create -f secret-pod.yaml
pod "test-projected-volume" created
```

6. Pod コンテナー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>
```

出力は以下のようになります。

```
$ oc get pod test-projected-volume
NAME                    READY     STATUS    RESTARTS   AGE
test-projected-volume   1/1       Running   0          14s
```

7. 別のターミナルで、**oc exec** コマンドを使用し、実行中のコンテナーに対してシェルを開きます。

```
$ oc exec -it <pod> <command>
```

例:

```
$ oc exec -it test-projected-volume -- /bin/sh
```

8. シェルで、**projected-volumes** ディレクトリーに展開されるソースが含まれることを確認します。

```
/ # ls
bin              home             root             tmp
dev              proc             run              usr
etc              projected-volume  sys              var
```

3.4. コンテナーによる API オブジェクト使用の許可

Downward API は、OpenShift Dedicated に結合せずにコンテナーが API オブジェクトについての情報を使用できるメカニズムです。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。コンテナーは、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。

3.4.1. Downward API の使用によるコンテナーへの 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。コンテナーは、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。

3.4.2. Downward API を使用してコンテナーの値を使用する方法について

コンテナーは、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、コンテナー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。

3.4.2.1. 環境変数の使用によるコンテナー値の使用

コンテナーの環境変数を使用する際に、**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

環境変数を使用するには、以下を実行します。

1. **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: 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
  restartPolicy: Never
```

2. **pod.yaml** ファイルから Pod を作成します。

```
$ oc create -f pod.yaml
```

3. コンテナーのログで **MY_POD_NAME** および **MY_POD_NAMESPACE** の値を確認します。

```
$ oc logs -p dapi-env-test-pod
```

3.4.2.2. ボリュームプラグインを使用したコンテナー値の使用

コンテナーは、ボリュームプラグイン使用して API 値を使用できます。

コンテナーは、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. **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:
  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
  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
```

2. **volume-pod.yaml** ファイルから Pod を作成します。

```
$ oc create -f volume-pod.yaml
```

3. コンテナーのログを確認し、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
cluster=downward-api-test-cluster1
rack=rack-123
zone=us-east-coast
annotation1=345
annotation2=456
kubernetes.io/config.source=api
```

3.4.3. Downward API を使用してコンテナー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。

3.4.3.1. 環境変数を使用したコンテナーリソースの使用

Pod を作成するときは、downward API を使用し、環境変数を使っ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。

手順

環境変数を使用するには、以下の手順を実行します。

1. 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。

```
....
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
....
```

リソース制限がコンテナー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。

2. **pod.yaml** ファイルから Pod を作成します。

```
$ oc create -f pod.yaml
```

3.4.3.2. ボリュームプラグインを使用したコンテナーリソースの使用

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っ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。

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. 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。

```
....
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
....
```

リソース制限がコンテナー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。

2. **volume-pod.yaml** ファイルから Pod を作成します。

```
$ oc create -f volume-pod.yaml
```

3.4.4. 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. **secret.yaml** ファイルを作成します。

```
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
data:
  password: cGFzc3dvcmQ=
  username: ZGV2ZWxvcGVy
type: kubernetes.io/basic-auth
```

2. secret.yaml ファイルから **Secret** を作成します。

```
$ oc create -f secret.yaml
```

3. 上記の **Secret** から **username** フィールドを参照する **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: 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
  restartPolicy: Never
```

4. **pod.yaml** ファイルから Pod を作成します。

```
$ oc create -f pod.yaml
```

5. コンテナーのログで **MY_SECRET_USERNAME** の値を確認します。

```
$ oc logs -p dapi-env-test-pod
```

3.4.5. 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. **configmap.yaml** ファイルを作成します。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: myconfigmap
data:
  mykey: myvalue
```

2. **configmap.yaml** ファイルから **ConfigMap** を作成します。

```
$ oc create -f configmap.yaml
```

3. 上記の **ConfigMap** を参照する **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: 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
  restartPolicy: Always
```

4. **pod.yaml** ファイルから Pod を作成します。

```
$ oc create -f pod.yaml
```

5. コンテナーのログで **MY_CONFIGMAP_VALUE** の値を確認します。

```
$ oc logs -p dapi-env-test-pod
```

3.4.6. 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 **environment variable** を参照する **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: 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)
  restartPolicy: Never
```

2. **pod.yaml** ファイルから Pod を作成します。

```
$ oc create -f pod.yaml
```

3. コンテナーのログで **MY_ENV_VAR_REF_ENV** 値を確認します。

```
$ oc logs -p dapi-env-test-pod
```

3.4.7. 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 **environment variable** を参照する **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: env-test-container
      image: gcr.io/google_containers/busybox
      command: [ "/bin/sh", "-c", "env" ]
      env:
        - name: MY_NEW_ENV
          value: $$(SOME_OTHER_ENV)
  restartPolicy: Never
```

2. **pod.yaml** ファイルから Pod を作成します。

```
$ oc create -f pod.yaml
```

3. コンテナーのログで **MY_NEW_ENV** 値を確認します。

```
$ oc logs -p dapi-env-test-pod
```

3.5. OPENSHIFT DEDICATED コンテナーへの/から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ー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、または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。

3.5.1. 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>]
```

3.5.1.1. 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 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。

3.5.2. コンテナーへの/からのファイルのコピー

コンテナーへの/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

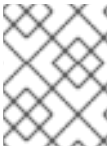
oc sync を使用する場合は、以下の点に注意してください。

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ー上に存在する場合は、ローカルの **rsync** ツールを使用します。

rsync がローカルの場所またはリモートコンテナー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ーに送信されます。ここで、**tar** ユーティリティーがファイルの展開に使用されます。 リモートコンテナーで **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 **rsync** と同様に機能する訳ではありません。たとえば、**rsync** は、宛先ディレクトリー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ーを Pod 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 <local-dir> <pod-name>:/<remote-dir>
```

例:

```
$ oc rsync /home/user/source devpod1234:/src

WARNING: cannot use rsync: rsync not available in container
status.txt
```

- 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

oc rsync devpod1234:/src/status.txt /home/user/
```

```
WARNING: cannot use rsync: rsync not available in container
status.txt
```

3.5.3. 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場合 (**--exclude-from=FILE** オプションなど)、以下のように標準の **rsync** の **--rsh** (**-e**) オプションまたは **RSYNC_RSH** 環境変数を回避策として使用することができます。

```
$ rsync --rsh='oc rsh' --exclude-from=FILE SRC POD:DEST
```

または、以下を実行します。

```
$ export RSYNC_RSH='oc rsh'
$ rsync --exclude-from=FILE SRC POD:DEST
```

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。

3.6. OPENSHIFT DEDICATED コンテナーでのリモートコマンドの実行

OpenShift Dedicated コンテナーでリモートコマンドを実行するために、CLI を使用することができます。

3.6.1. コンテナーでのリモートコマンドの実行

リモートコンテナー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

コンテナー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>]
```

例:

```
$ oc exec mypod date
Thu Apr  9 02:21:53 UTC 2015
```

重要

セキュリティー保護の理由により、**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、特権付きコンテナーにアクセスしようとしても機能しません。

3.6.2. 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ー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ーの名前です。
- **<command>** は実行される必要なコマンドです。

例:

```
/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。
- リモートコンテナー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。
- リモートコンテナー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 **SPDY** 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。

3.7. コンテナー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

OpenShift Dedicated は Pod へのポート転送をサポートします。

3.7.1. 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の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。

OpenShift Dedicated 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、OpenShift Dedicated 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。OpenShift Dedicated が新規ストリームを受信したら、ストリームと Pod のポート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 OpenShift Dedicated 実装はノードホストで **nsenter** を直接呼び出して、Pod ネットワークの 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。

3.7.2. 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>]
```

例:

- 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

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

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
```

```
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
```

3.7.3. 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。

例:

```
/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 SPDY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。

3.8. コンテナーでの SYSCTL の使用

sysctl 設定は Kubernetes 経由で公開され、ユーザーがコンテナー内の namespace の特定のカーネルパラメーターをランタイム時に変更できるようにします。namespace を使用する sysctl のみを Pod 上で独立して設定できます。sysctl に namespace が使用されていない場合 (この状態は **ノードレベル** と呼ばれる)、OpenShift Dedicated 内でこれを設定することはできません。さらに **安全** とみなされる sysctl のみがデフォルトでホワイトリスト化されます。 他の **安全でない** sysctl はノードで手動で有効にし、ユーザーが使用できるようにできます。

3.8.1. sysctl について

Linux では、管理者は sysctl インターフェースを使ってランタイム時にカーネルパラメーターを変更することができます。パラメーターは **/proc/sys/** 仮想プロセスファイルシステムで利用できます。これらのパラメーターは以下を含む各種のサブシステムを対象とします。

- カーネル (共通のプレフィックス: **kernel.**)
- ネットワーク (共通のプレフィックス: **net.**)

- 仮想メモリー (共通のプレフィックス: **vm.**)
- MDADM (共通のプレフィックス: **dev.**)

追加のサブシステムについては、カーネルのドキュメントで説明されています。すべてのパラメーターの一覧を表示するには、以下のコマンドを実行します。

```
$ sudo sysctl -a
```

3.8.1.1. namespace を使用した sysctl vs ノードレベルの sysctl

Linux カーネルでは、数多くの sysctl に **namespace** が使用されています。これは、それらをノードの各 Pod に対して個別に設定できることを意味します。namespace の使用は、sysctl を Kubernetes 内の Pod 環境でアクセス可能にするための要件になります。

以下の sysctl は namespace を使用するものとして知られている sysctl です。

- **kernel.shm***
- **kernel.msg***
- **kernel.sem**
- **fs.mqueue.***

また、**net.*** グループの大半の sysctl には namespace が使用されていることが知られています。それらの namespace の使用は、カーネルのバージョンおよびディストリビューターによって異なります。

namespace が使用されていない sysctl は **ノードレベル** と呼ばれており、クラスター管理者がノードの基礎となる Linux ディストリビューションを使用 (例: **/etc/sysctls.conf** ファイルを変更) するか、または特権付きコンテナーで DaemonSet を使用することによって手動で設定する必要があります。

注記

特殊な sysctl が設定されたノードにテイントのマークを付けることを検討してください。それらの sysctl 設定を必要とするノードにのみ Pod をスケジュールします。テイントおよび容認 (Toleration) 機能を使用してノードにマークを付けます。

3.8.1.2. 安全 vs 安全でない sysctl

sysctl は **安全な** および **安全でない** sysctl に分類されます。

sysctl が安全であるとみなされるには、適切な namespace を使用し、同じノード上の Pod 間で適切に分離する必要があります。Pod ごとに sysctl を設定する場合は、以下の点に留意してください。

- この設定はノードのその他の Pod に影響を与えないものである。
- この設定はノードの正常性に負の影響を与えないものである。
- この設定は Pod のリソース制限を超える CPU またはメモリーリソースの取得を許可しないものである。

OpenShift Dedicated は以下の sysctl を安全なセットでサポートするか、またはホワイトリスト化します。

- **kernel.shm_rmid_forced**

- net.ipv4.ip_local_port_range
- net.ipv4.tcp_syncookies

すべての安全な sysctl はデフォルトで有効にされます。Pod 仕様を変更して、Pod で sysctl を使用できます。

OpenShift Dedicated でホワイトリスト化されない sysctl は OpenShift Dedicated で安全でないと見なされます。namespace を使用するだけで、sysctl が安全であるとみなされる訳ではありません。

すべての安全でない sysctl はデフォルトで無効にされ、ノードごとにクラスター管理者によって手動で有効にされる必要があります。無効にされた安全でない sysctl が設定された Pod はスケジュールされますが、起動されません。

```
$ oc get pod

NAME       READY   STATUS          RESTARTS   AGE
hello-pod  0/1     SysctlForbidden  0         14s
```

3.8.2. Pod の sysctl 設定

Pod の **securityContext**を使用して sysctl を Pod に設定できます。 **securityContext** は同じ Pod 内のすべてのコンテナーに適用されます。

安全な sysctl はデフォルトで許可されます。安全でない sysctl が設定された Pod は、クラスター管理者がそのノードの安全でない sysctl を明示的に有効にしない限り、いずれのノードでも起動に失敗します。ノードレベルの sysctl の場合のように、それらの Pod を正しいノードにスケジュールするには、テイントおよび容認 (Toleration)、またはノードのラベルを使用します。

以下の例では、Pod の **securityContext** を使用して安全な sysctl **kernel.shm_rmid_forced** および 2 つの安全でない sysctl **net.ipv4.route.min_pmtu** および **kernel.msgmax** を設定します。仕様では **安全な** sysctl と **安全でない** sysctl は区別されません。

警告

オペレーティングシステムが不安定になるのを防ぐには、変更の影響を確認している場合にのみ sysctl パラメーターを変更します。

手順

安全なおよび安全でない sysctl を使用するには、以下を実行します。

1. 以下の例に示されるように、Pod を定義する YAML ファイルを変更し、**securityContext** 仕様を追加します。

```
apiVersion: v1
kind: Pod
metadata:
  name: sysctl-example
spec:
  securityContext:
```

```
    sysctls:
    - name: kernel.shm_rmid_forced
      value: "0"
    - name: net.ipv4.route.min_pmtu
      value: "552"
    - name: kernel.msgmax
      value: "65536"
  ...
```

2. Pod を作成します。

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

安全でない sysctl がノードに許可されていない場合、Pod はスケジュールされますが、デプロイはされません。

```
$ oc get pod

NAME        READY   STATUS            RESTARTS   AGE
hello-pod   0/1     SysctlForbidden   0          14s
```

3.8.3. 安全でない sysctl の有効化

クラスター管理者は、高パフォーマンスまたはリアルタイムのアプリケーション調整などの非常に特殊な状況で特定の安全でない sysctl を許可することができます。

安全でない sysctl を使用する必要がある場合、クラスター管理者は特定のタイプのノードに対してそれらを個別に有効にする必要があります。sysctl には namespace を使用する必要があります。

警告

安全でないという性質上、安全でない sysctl は各自の責任で使用されます。 場合によっては、コンテナーの正しくない動作やリソース不足、またはノードの破損などの深刻な問題が生じる可能性があります。

手順

1. ラベルを安全でない sysctl が設定されたコンテナーが実行される MachineConfigPool に追加します。

```
$ oc edit machineconfigpool worker
```

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  creationTimestamp: 2019-02-08T14:52:39Z
  generation: 1
  labels:
    custom-kubelet: sysctl 1
```

1 **key: pair** ラベルを追加します。

2. KubeletConfig カスタムリソース (CR) を作成します。

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: custom-kubelet
spec:
  machineConfigPoolSelector:
    matchLabels:
      custom-kubelet: sysctl 1
  kubeletConfig:
    allowedUnsafeSysctls: 2
      - "kernel.msg*"
      - "net.ipv4.route.min_pmtu"
```

1 MachineConfigPool からラベルを指定します。

2 許可する必要のある安全でない sysctl を一覧表示します。

3. オブジェクトを作成します。

```
$ oc apply -f set-sysctl-worker.yaml
```

99-worker-XXXXXXXXX-xxxxx-XXXX-xxxxx-kubelet 形式の新規の MachineConfig が作成されます。

4. **machineconfigpool** オブジェクトの **status** フィールドを使用してクラスターが再起動するのを待機します。
例:

```
status:
  conditions:
    - lastTransitionTime: '2019-08-11T15:32:00Z'
      message: >-
        All nodes are updating to
        rendered-worker-ccbfb5d2838d65013ab36300b7b3dc13
      reason: ''
      status: 'True'
      type: Updating
```

クラスターの準備ができると、以下のようなメッセージが表示されます。

```
    - lastTransitionTime: '2019-08-11T16:00:00Z'
      message: >-
        All nodes are updated with
        rendered-worker-ccbfb5d2838d65013ab36300b7b3dc13
      reason: ''
      status: 'True'
      type: Updated
```

5. クラスターが準備状態になる場合、新規 MachineConfig でマージされた **KubeletConfig** を確認します。

```
$ oc get machineconfig 99-worker-XXXXXX-XXXXX-XXXX-XXXXX-kubelet -o json | grep ownerReference -A7
```

```
        "ownerReferences": [
            {
                "apiVersion": "machineconfiguration.openshift.io/v1",
                "blockOwnerDeletion": true,
                "controller": true,
                "kind": "KubeletConfig",
                "name": "custom-kubelet",
                "uid": "3f64a766-bae8-11e9-abe8-0a1a2a4813f2"
```

安全でない sysctl を必要に応じて Pod に追加することができるようになります。

第4章 クラスターの操作

4.1. OPENSHIFT DEDICATED クラスターでのシステムイベント情報の表示

OpenShift Dedicated のイベントは、OpenShift Dedicated クラスターの API オブジェクトに対して発生するイベントに基づいてモデル化されます。

4.1.1. イベントについて

イベントにより、OpenShift Dedicated 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。

4.1.2. 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一覧を取得できます。

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。

例:

```
$ oc get events -n openshift-config

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(748c7a40db3d08c07fb4f9eba774bd5effe5f0d5090a242432a73eee66ba9e22
): 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
```

- OpenShift Dedicated 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. OpenShift Dedicated 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。

4.1.3. イベントの一覧

このセクションでは、OpenShift Dedicated のイベントについて説明します。

表4.1 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4.2 コンテナーイベント

名前	説明
BackOff	バックオフ (再起動) によりコンテナーが失敗しました。
Created	コンテナーが作成されました。
Failed	プル/作成/起動が失敗しました。
Killing	コンテナーを強制終了しています。
Started	コンテナーが起動しました。
Preempting	他の Pod を退避します。
ExceededGracePeriod	コンテナーランタイムは、指定の猶予期間以内に Pod を停止しませんでした。

表4.3 正常性イベント

名前	説明
Unhealthy	コンテナーが正常ではありません。

表4.4 イメージイベント

名前	説明
BackOff	バックオフ (コンテナー起動、イメージのプル)。
ErrImageNeverPull	イメージの **NeverPull Policy** の違反があります。

名前	説明
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、コンテナー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4.5 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。
InvalidDiskCapacity	無効なディスク容量です。

表4.6 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
InsufficientFreeCPU	空き CPU が十分にありません。
InsufficientFreeMemory	空きメモリーが十分にありません。
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。

名前	説明
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。
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。

名前	説明
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	コンテナー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。
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4.7 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4.8 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4.9 Pod イベント

名前	説明
FailedKillPod	Pod の停止に失敗しました。
FailedCreatePodContainer	Pod コンテナー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>**.
SuccessfulDelete	削除した Pod: **<pod-id>**.

表4.10 Horizontal Pod AutoScaler 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リング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。

名前	説明
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4.11 ネットワークイベント (openshift-sdn)

名前	説明
Starting	OpenShift-SDN を開始します。
NetworkFailed	Pod のネットワークインターフェースがなくなり、Pod が停止します。

表4.12 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4.13 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。

名前	説明
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4.14 ライフサイクルフック

名前	説明
FailedPostStartHook	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4.15 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4.16 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4.17 DaemonSet 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。
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4.18 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>**

名前	説明
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-ploicy>**
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。
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。