OpenShift Container Platform 3.11

クラスター管理

OpenShift Container Platform 3.11 クラスター管理

Last Updated: 2023-03-25

OpenShift Container Platform 3.11 クラスター管理

OpenShift Container Platform 3.11 クラスター管理

Enter your first name here. Enter your surname here.
Enter your organisation's name here. Enter your organisational division here.
Enter your email address here.

法律上の通知

概要

OpenShift クラスター管理では、OpenShift クラスターを管理するための通常のタスクや他の詳細設定についてのトピックを扱います。

目次

第1章 概要

OpenShift クラスター管理では、OpenShift Container Platform クラスターを管理するための日常的なタスクや他の詳細な設定についてのトピックを扱います。

第2章 ノードの管理

2.1. 概要

インスタンスの ノード は、CLI を使用して管理することができます。

ノードの管理操作を実行する際、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、ヘルスチェック でノードの検証を実行します。

2.2. ノードの一覧表示

マスターに認識されるすべてのノードを一覧表示するには、以下を実行します。

```
$ oc get nodes
```

出力例

```
NAME                STATUS    ROLES     AGE       VERSION
master.example.com  Ready     master    7h        v1.9.1+a0ce1bc657
node1.example.com   Ready     compute   7h        v1.9.1+a0ce1bc657
node2.example.com   Ready     compute   7h        v1.9.1+a0ce1bc657
```

ノード情報を含むプロジェクトの Pod デプロイメントの情報と共にすべてのノードを一覧表示するには、以下を実行します。

```
$ oc get nodes -o wide
```

出力例

```
NAME                         STATUS    ROLES     AGE       VERSION         EXTERNAL-IP     OS-IMAGE
KERNEL-VERSION          CONTAINER-RUNTIME
ip-172-18-0-39.ec2.internal  Ready     infra     1d        v1.10.0+b81c8f8   54.172.185.130   Red Hat
Enterprise Linux Server 7.5 (Maipo)   3.10.0-862.el7.x86_64   docker://1.13.1
ip-172-18-10-95.ec2.internal  Ready     master    1d        v1.10.0+b81c8f8   54.88.22.81      Red Hat
Enterprise Linux Server 7.5 (Maipo)   3.10.0-862.el7.x86_64   docker://1.13.1
ip-172-18-8-35.ec2.internal  Ready     compute   1d        v1.10.0+b81c8f8   34.230.50.57     Red Hat
Enterprise Linux Server 7.5 (Maipo)   3.10.0-862.el7.x86_64   docker://1.13.1
```

単一ノードに関する情報のみを一覧表示するには、**<node>** を完全なノード名に置き換えます。

```
$ oc get node <node>
```

これらのコマンドの出力にある **STATUS** 列には、ノードの以下の状態が表示されます。

表2.1 ノードの状態

条件	説明
Ready	ノードは **StatusOK** を返し、マスターから実行されるヘルスチェックをパスしています。

条件	説明
NotReady	ノードはマスターから実行されるヘルスチェックをパスしていません。
SchedulingDisabled	ノードへの Pod の 配置をスケジュール できません。

注記

STATUS 列には、CLI でノードの状態を検索できない場合にノードについて **Unknown** が表示されます。

現在の状態の理由を含む特定ノードに関する詳細情報を取得するには、以下を実行します。

```
$ 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:              compute 2
Labels:             beta.kubernetes.io/arch=amd64 3
                    beta.kubernetes.io/os=linux
                    kubernetes.io/hostname=m01.example.com
                    node-role.kubernetes.io/compute=true
                    node-role.kubernetes.io/infra=true
                    node-role.kubernetes.io/master=true
                    zone=default
Annotations:        volumes.kubernetes.io/controller-managed-attach-detach=true 4
CreationTimestamp:  Thu, 24 May 2018 11:46:56 -0400
Taints:             <none> 5
Unschedulable:      false
Conditions:                 6
  Type             Status  LastHeartbeatTime                 LastTransitionTime                Reason
Message
  ----             ------  -----------------                 ------------------                ------
  OutOfDisk        False   Tue, 17 Jul 2018 11:47:30 -0400   Tue, 10 Jul 2018 15:45:16 -0400
KubeletHasSufficientDisk     kubelet has sufficient disk space available
  MemoryPressure   False   Tue, 17 Jul 2018 11:47:30 -0400   Tue, 10 Jul 2018 15:45:16 -0400
KubeletHasSufficientMemory   kubelet has sufficient memory available
  DiskPressure     False   Tue, 17 Jul 2018 11:47:30 -0400   Tue, 10 Jul 2018 16:03:54 -0400
KubeletHasNoDiskPressure     kubelet has no disk pressure
  Ready            True    Tue, 17 Jul 2018 11:47:30 -0400   Mon, 16 Jul 2018 15:10:25 -0400
KubeletReady                 kubelet is posting ready status
  PIDPressure      False   Tue, 17 Jul 2018 11:47:30 -0400   Thu, 05 Jul 2018 10:06:51 -0400
KubeletHasSufficientPID      kubelet has sufficient PID available
```

```
Addresses:                  7
  InternalIP:  192.168.122.248
  Hostname:    node1.example.com
Capacity:                   8
 cpu:           2
 hugepages-2Mi:  0
 memory:        8010336Ki
 pods:          40
Allocatable:
 cpu:           2
 hugepages-2Mi:  0
 memory:        7907936Ki
 pods:          40
System Info:                9
 Machine ID:                 b3adb9acbc49fc1f9a7d6
 System UUID:                B3ADB9A-B0CB-C49FC1F9A7D6
 Boot ID:                    9359d15aec9-81a20aef5876
 Kernel Version:             3.10.0-693.21.1.el7.x86_64
 OS Image:                   OpenShift Enterprise
 Operating System:           linux
 Architecture:               amd64
 Container Runtime Version:  docker://1.13.1
 Kubelet Version:            v1.10.0+b81c8f8
 Kube-Proxy Version:         v1.10.0+b81c8f8
ExternalID:                  node1.example.com
Non-terminated Pods:         (14 in total)   10
  Namespace                  Name                                CPU Requests  CPU Limits  Memory
Requests  Memory Limits
  ---------                  ----                                ------------  ----------  ---------------  -------------
  default                    docker-registry-2-w252l             100m (5%)     0 (0%)      256Mi (3%)       0
(0%)
  default                    registry-console-2-dpnc9            0 (0%)        0 (0%)      0 (0%)           0 (0%)
  default                    router-2-5snb2                      100m (5%)     0 (0%)      256Mi (3%)       0
(0%)
  kube-service-catalog       apiserver-jh6gt                     0 (0%)        0 (0%)      0 (0%)           0
(0%)
  kube-service-catalog       controller-manager-z4t5j            0 (0%)        0 (0%)      0 (0%)           0
(0%)
  kube-system                master-api-m01.example.com          0 (0%)        0 (0%)      0 (0%)
0 (0%)
  kube-system                master-controllers-m01.example.com  0 (0%)        0 (0%)      0 (0%)
0 (0%)
  kube-system                master-etcd-m01.example.com         0 (0%)        0 (0%)      0 (0%)
0 (0%)
  openshift-ansible-service-broker  asb-1-hnn5t                  0 (0%)        0 (0%)      0 (0%)           0
(0%)
  openshift-node             sync-dvhvs                          0 (0%)        0 (0%)      0 (0%)           0 (0%)
  openshift-sdn              ovs-zjs5k                           100m (5%)     200m (10%)  300Mi (3%)
400Mi (5%)
  openshift-sdn              sdn-zr4cb                           100m (5%)     0 (0%)      200Mi (2%)       0
(0%)
  openshift-template-service-broker  apiserver-s9n7t             0 (0%)        0 (0%)      0 (0%)
0 (0%)
  openshift-web-console      webconsole-785689b664-q7s9j         100m (5%)     0 (0%)      100Mi
(1%)       0 (0%)
Allocated resources:
```

```
  (Total limits may be over 100 percent, i.e., overcommitted.)
  CPU Requests  CPU Limits  Memory Requests  Memory Limits
  ------------  ----------  ---------------  -------------
  500m (25%)    200m (10%)  1112Mi (14%)     400Mi (5%)
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**、**compute**、または **infra** のいずれか。
3. ノードに適用される ラベル。
4. ノードに適用されるアノテーション。
5. ノードに適用される テイント。
6. ノードの状態。
7. ノードの IP アドレスおよびホスト名。
8. Pod リソースおよび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. ノードで報告される イベント。

2.3. ノードの表示

コンテナー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。

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%
node-1     297m         29%       4263Mi          55%
node-0     55m          5%        1201Mi          15%
infra-1    85m          8%        1319Mi          17%
infra-0    182m         18%       2524Mi          32%
master-0   178m         8%        2584Mi          16%
```

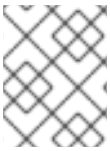
ラベルの付いたノードの使用状況の統計を表示するには、以下を実行します。

```
$ 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。

注記

使用状況の統計を閲覧するには、**cluster-reader** パーミッションがなければなりません。

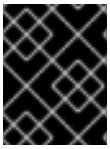


注記

使用状況の統計を閲覧するには、**metrics-server** がインストールされている必要があります。Horizontal Pod Autoscaler の要件 を参照してください。

2.4. ホストの追加

scaleup.yml Playbook を実行して新規ホストをクラスターに追加できます。この Playbook はマスターをクエリーし、新規ホストの新規証明書を生成し、配布してから、設定 Playbook を新規ホストにのみ実行します。**scaleup.yml** Playbook を実行する前に、前提条件となる ホストの準備 手順をすべて完了してください。



重要

scaleup.yml の Playbook は新規ホストの設定のみを設定します。マスターサービスの **NO_PROXY** の更新やマスターサービスの再起動は行いません。

scaleup.yml Playbook を実行するには、現在のクラスター設定を表す既存のインベントリーファイル (**/etc/ansible/hosts** など) が必要です。以前に **atomic-openshift-installer** コマンドを使用してインストールを実行した場合は、**~/.config/openshift/hosts** を調べて、インストーラーによって生成された最新のインベントリーファイルを見つけ、そのファイルをインベントリーファイルとして使用することができます。このファイルは必要に応じて変更することができます。後で **ansible-playbook** を実行する際に **-i** を使用して、そのファイルの場所を指定する必要があります。

重要

ノードの推奨の最大数については、クラスターの最大値 のセクションを参照してください。

手順

1. **openshift-ansible** パッケージを更新して最新の Playbook を取得します。

```
# yum update openshift-ansible
```

2. **/etc/ansible/hosts** ファイルを編集し、**new_<host_type>** を **[OSEv3:children]** セクションに追加します。たとえば、新規ノードホストを追加するには、**new_nodes** を追加します。

```
[OSEv3:children]
masters
nodes
new_nodes
```

新規マスターホストを追加するには、**new_masters** を追加します。

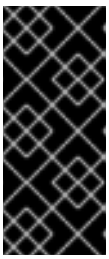
3. **[new_<host_type>]** セクションを作成して、新規ホストのホスト情報を指定します。以下の新規ノードの追加例で示されているように、既存のセクションと同じ様にこのセクションをフォーマットします。

```
[nodes]
master[1:3].example.com
node1.example.com openshift_node_group_name='node-config-compute'
node2.example.com openshift_node_group_name='node-config-compute'
infra-node1.example.com openshift_node_group_name='node-config-infra'
infra-node2.example.com openshift_node_group_name='node-config-infra'

[new_nodes]
node3.example.com openshift_node_group_name='node-config-infra'
```

その他のオプションについては、ホスト変数の設定 を参照してください。

新規マスターを追加する場合は、**[new_masters]** セクションと **[new_nodes]** セクションの両方にホストを追加して、新規マスターホストが OpenShift SDN の一部となるようにします。

```
[masters]
master[1:2].example.com

[new_masters]
master3.example.com

[nodes]
master[1:2].example.com
node1.example.com openshift_node_group_name='node-config-compute'
node2.example.com openshift_node_group_name='node-config-compute'
infra-node1.example.com openshift_node_group_name='node-config-infra'
infra-node2.example.com openshift_node_group_name='node-config-infra'

[new_nodes]
master3.example.com
```



重要

マスターホストに **node-role.kubernetes.io/infra=true** ラベルを付け、それ以外に専用インフラストラクチャーノードがない場合は、エントリーに **openshift_schedulable=true** を追加してホストにスケジュール可能であることを示すマークを明示的に付ける必要もあります。そうしないと、レジストリー Pod とルーター Pod をどこにも配置できなくなります。

4. Playbook ディレクトリーに切り替え、**openshift_node_group.yml** Playbook を実行します。インベントリーファイルがデフォルトの **/etc/ansible/hosts** 以外の場所にある場合は、**-i** オプションで場所を指定します。

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook [-i /path/to/file] \
  playbooks/openshift-master/openshift_node_group.yml
```

これにより、新規ノードグループの ConfigMap が作成され、最終的にホスト上のノードの設定ファイルが作成されます。

注記

openshift_node_group.yaml Playbook を実行すると、新規ノードのみが更新されます。クラスター内の既存ノードを更新するために実行することはできません。

5. **scaleup.yml** Playbook を実行します。インベントリーファイルがデフォルトの **/etc/ansible/hosts** 以外の場所にある場合は、**-i** オプションで場所を指定します。

- ノードを追加する場合は、以下を指定します。

```
$ ansible-playbook [-i /path/to/file] \
    playbooks/openshift-node/scaleup.yml
```

- マスターを追加する場合は、以下を実行します。

```
$ ansible-playbook [-i /path/to/file] \
    playbooks/openshift-master/scaleup.yml
```

6. EFK スタックをクラスターにデプロイしている場合は、ノードラベルを **logging-infra-fluentd=true** に設定します。

```
# oc label node/new-node.example.com logging-infra-fluentd=true
```

7. Playbook の実行後に、インストールの検証 を行います。

8. **[new_<host_type>]** セクションで定義したホストを適切なセクションに移動します。このようにホストを移動することで、このインベントリーファイルを使用するその後の Playbook の実行で、正しくノードが処理されるようになります。**[new_<host_type>]** セクションは空のままにできます。たとえば、新規ノードを追加する場合は、以下のように指定します。

```
[nodes]
master[1:3].example.com
node1.example.com openshift_node_group_name='node-config-compute'
node2.example.com openshift_node_group_name='node-config-compute'
node3.example.com openshift_node_group_name='node-config-compute'
infra-node1.example.com openshift_node_group_name='node-config-infra'
infra-node2.example.com openshift_node_group_name='node-config-infra'

[new_nodes]
```

2.5. ノードの削除

CLI を使用してノードを削除する場合、ノードオブジェクトは Kubernetes で削除されますが、ノード自体にある Pod は削除されません。レプリケーションコントローラーでサポートされないベア Pod は OpenShift Container Platform からアクセスできなくなり、レプリケーションコントローラーでサポートされる Pod は他の利用可能なノードにスケジュール変更されます。ローカルのマニフェスト Pod は手動で削除する必要があります。

OpenShift Container Platform クラスターからノードを削除するには、以下を実行します。

1. 削除しようとしているノードから Pod を退避 します。
2. ノードオブジェクトを削除します。

```
$ oc delete node <node>
```

3. ノードがノード一覧から削除されていることを確認します。

```
$ oc get nodes
```

Pod は、 **Ready** 状態にある残りのノードに対してのみスケジュールされる必要があります。

4. すべての Pod およびコンテナーを含む OpenShift Container Platform のすべてのコンテンツをノードホストからアンインストールする必要がある場合は、ノードのアンインストール を継続し、**uninstall.yml** Playbook を使用する手順に従います。この手順では、Ansible を使用した クラスターインストールプロセス についての全般的な理解があることが前提となります。

2.6. ノードのラベルの更新

ノードで ラベル を追加または更新するには、以下を実行します。

```
$ oc label node <node> <key_1>=<value_1> ... <key_n>=<value_n>
```

詳細な使用法を表示するには、以下を実行します。

```
$ oc label -h
```

2.7. ノード上の POD の一覧表示

1 つ以上のノードに、すべてまたは選択した Pod を一覧表示するには、以下を実行します。

```
$ oc adm manage-node <node1> <node2> \
    --list-pods [--pod-selector=<pod_selector>] [-o json|yaml]
```

選択したノードに、すべてまたは選択した Pod を一覧表示するには、以下を実行します。

```
$ oc adm manage-node --selector=<node_selector> \
    --list-pods [--pod-selector=<pod_selector>] [-o json|yaml]
```

2.8. ノードをスケジュール対象外 (UNSCHEDULABLE) またはスケジュール対象 (SCHEDULABLE) としてマークする

デフォルトで、**Ready** ステータス の正常なノードはスケジュール対象としてマークされます。つま

り、新規 Pod をこのノードに配置することができます。手動でノードをスケジュール対象外としてマークすると、新規 Pod のノードでのスケジュールがブロックされます。ノード上の既存 Pod には影響がありません。

1 つまたは複数のノードをスケジュール対象外としてマークするには、以下を実行します。

```
$ oc adm manage-node <node1> <node2> --schedulable=false
```

以下に例を示します。

```
$ oc adm manage-node node1.example.com --schedulable=false
```

出力例

```
NAME                LABELS                                        STATUS
node1.example.com   kubernetes.io/hostname=node1.example.com      Ready,SchedulingDisabled
```

現時点でスケジュール対象外のノードをスケジュール対象としてマークするには、以下を実行します。

```
$ oc adm manage-node <node1> <node2> --schedulable
```

または、特定のノード名 (例: **<node1> <node2>**) を指定する代わりに、**--selector=<node_selector>** オプションを使用して選択したノードをスケジュール対象またはスケジュール対象外としてマークすることができます。

2.9. POD のノードからの退避

Pod を退避することで、すべての Pod または選択した Pod を 1 つの指定されたノードから移行できます。Pod の退避を実行するには、まずノードを スケジュール対象外としてマーク する必要があります。

レプリケーションコントローラー でサポートされる Pod のみを退避できます。レプリケーションコントローラーは他のノードで新規 Pod を作成し、既存の Pod を指定したノードから削除します。デフォルトで、ベア Pod (レプリケーションコントローラーでサポートされていない Pod) はこの影響を受けません。Pod セレクターを指定すると Pod のサブセットを退避できます。Pod セレクターはラベルをベースにしており、指定されたラベルのあるすべての Pod の退避が行われます。

ノードですべての Pod または選択した Pod を退避するには、以下を実行します。

```
$ oc adm drain <node> [--pod-selector=<pod_selector>]
```

--force オプションを使用すると、ベア Pod の削除を強制的に実行できます。**true** に設定されると、Pod がレプリケーションコントローラー、ReplicaSet、ジョブ、DeamonSet、または StatefulSet で管理されていない場合でも削除が続行されます。

```
$ oc adm drain <node> --force=true
```

--grace-period を使用して、各 Pod を正常に終了するための期間 (秒単位) を設定できます。負の値の場合には、Pod に指定されるデフォルト値が使用されます。

```
$ oc adm drain <node> --grace-period=-1
```

--ignore-daemonsets を使用し、これを **true** に設定すると、Deamonset で管理された Pod を無視できます。

```
$ oc adm drain <node> --ignore-daemonsets=true
```

--timeout を使用すると、中止する前の待機期間を設定できます。値 **0** は無限の時間を設定します。

```
$ oc adm drain <node> --timeout=5s
```

--delete-local-data を使用し、これを **true** に設定すると、Pod が emptyDir (ノードがドレイン (解放) される場合に削除されるローカルデータ) を使用する場合でも削除が続行されます。

```
$ oc adm drain <node> --delete-local-data=true
```

退避を実行せずに移行するオブジェクトを一覧表示するには、**--dry-run** オプションを使用し、これを **true** に設定します。

```
$ oc adm drain <node> --dry-run=true
```

特定のノード名を指定する代わりに、**--selector=<node_selector>** オプションを使用し、セレクターに一致するノードの Pod を退避することができます。

2.10. ノードの再起動

プラットフォームで実行されるアプリケーションを停止せずにノードを再起動するには、まず Pod の退避を実行する必要があります。ルーティング階層によって可用性が高くされている Pod については、何も実行する必要はありません。ストレージ (通常はデータベース) を必要とするその他の Pod については、それらが 1 つの Pod が一時的にオフラインになっても作動したままになることを確認する必要があります。ステートフルな Pod の回復性はアプリケーションごとに異なりますが、いずれの場合でも、ノードの非アフィニティー (node anti-affinity) を使用して Pod が使用可能なノード間に適切に分散するようにスケジューラーを設定することが重要になります。

別の課題として、ルーターやレジストリーのような重要なインフラストラクチャーを実行しているノードを処理する方法を検討する必要があります。同じノードの退避プロセスが適用されますが、一部のエッジケースについて理解しておくことが重要です。

2.10.1. インフラストラクチャーノード

インフラストラクチャーノードは、OpenShift Container Platform 環境の一部を実行するためにラベルが付けられたノードです。現在、ノードの再起動を管理する最も簡単な方法として、インフラストラクチャーを実行するために利用できる 3 つ以上のノードを確保することができます。以下のシナリオでは、2 つのノードのみが利用可能な場合に OpenShift Container Platform で実行されるアプリケーションのサービスを中断しかねないよくある問題を示しています。

- ノード A がスケジュール対象外としてマークされており、すべての Pod の退避が行われている。
- このノードで実行されているレジストリー Pod がノード B に再デプロイされる。 これは、ノード B が両方のレジストリー Pod を実行していることを意味します。
- ノード B はスケジュール対象外としてマークされ、退避が行われる。
- ノード B の 2 つの Pod エンドポイントを公開するサービスは、それらがノード A に再デプロイされるまでの短い期間すべてのエンドポイントを失う。

3 つのインフラストラクチャーノードを使用する同じプロセスではサービスの中断が生じません。しかし、Pod のスケジューリングにより、退避してローテーションに戻された最後のノードはゼロ (0) レジストリーを実行していることになり、他の 2 つのノードは 2 つのレジストリーと 1 つのレジストリーをそれぞれ実行します。他の 2 つのノードは 2 つのレジストリーと 1 つのレジストリーをそれぞれ実行します。最適なソリューションとして、Pod の非アフィニティーを使用します。これは現在テスト目的で利用できる Kubernetes のアルファ機能ですが、実稼働ワークロードに対する使用はサポートされていません。

2.10.2. Pod の非アフィニティーの使用

Pod の非アフィニティー は、ノードの非アフィニティー とは若干異なります。ノードの非アフィニティーの場合、Pod のデプロイ先となる適切な場所がほかにない場合には違反が生じます。Pod の非アフィニティーの場合は required (必須) または preferred (優先) のいずれかに設定できます。

```
apiVersion: v1
kind: Pod
metadata:
  name: with-pod-antiaffinity
spec:
  affinity:
    podAntiAffinity: 1
      preferredDuringSchedulingIgnoredDuringExecution: 2
      - weight: 100 3
        podAffinityTerm:
          labelSelector:
            matchExpressions:
            - key: docker-registry 4
              operator: In 5
              values:
              - default
          topologyKey: kubernetes.io/hostname
```

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。

この例では、コンテナーイメージレジストリー Pod に **docker-registry=default** のラベルがあることを想定しています。Pod の非アフィニティーでは任意の Kubernetes の一致式を使用できます。

最後に必要となる手順として、**/etc/origin/master/scheduler.json** で **MatchInterPodAffinity** スケジューラーの述語を有効にします。これが有効になっていると、2 つのインフラストラクチャーノードのみが利用可能で、1 つのノードが再起動される場合に、コンテナーイメージレジストリー Pod は他のノードで実行できなくなります。**oc get pods** は、適切なノードが利用可能になるまで Pod を Unready (準備が未完了) として報告します。ノードが利用可能になり、すべての Pod が Ready (準備ができている) 状態に戻ると、次のノードを再起動することができます。

2.10.3. ルーターを実行するノードの処理

ほとんどの場合、OpenShift Container Platform ルーターを実行する Pod はホストのポートを公開します。**PodFitsPorts** スケジューラーの述語により、同じポートを使用するルーター Pod が同じノードで実行されないようにし、Pod の非アフィニティーが適用されます。ルーターの高可用性を維持するために IP フェイルオーバー を利用している場合には、他に実行することはありません。高可用性を確保するために AWS Elastic Load Balancing などの外部サービスを使用するルーター Pod の場合は、そのような外部サービスがルーター Pod の再起動に対して対応します。

ルーター Pod でホストのポートが設定されていないということも稀にあります。この場合は、インフラストラクチャーノードについての 推奨される再起動プロセス に従う必要があります。

2.11. ノードの変更

インストール時に、OpenShift Container Platform はそれぞれのノードグループに対して **openshift-node** プロジェクトに configmap を作成します。

- node-config-master
- node-config-infra
- node-config-compute
- node-config-all-in-one
- node-config-master-infra

既存のノードに設定の変更を加えるには、該当する設定マップを編集します。各ノードの **sync pod** は設定マップで変更の有無を監視します。インストール時に、同期 Pod は **sync Daemonsets** を使用して作成され、ノード設定パラメーターが存在する **/etc/origin/node/node-config.yaml** ファイルが各ノードに追加されます。同期 Pod が設定マップの変更を検出すると、そのノードグループ内のすべてのノードで **node-config.yaml** を更新し、適切なノードで **atomic-openshift-node.service** を再起動します。

```
$ oc get cm -n openshift-node
```

出力例

```
NAME                   DATA      AGE
node-config-all-in-one     1         1d
node-config-compute        1         1d
node-config-infra          1         1d
node-config-master         1         1d
node-config-master-infra   1         1d
```

node-config-compute グループの設定マップの例

```
apiVersion: v1
authConfig:     (1)
  authenticationCacheSize: 1000
  authenticationCacheTTL: 5m
  authorizationCacheSize: 1000
  authorizationCacheTTL: 5m
dnsBindAddress: 127.0.0.1:53
```

```
dnsDomain: cluster.local
dnsIP: 0.0.0.0                                                    (2)
dnsNameservers: null
dnsRecursiveResolvConf: /etc/origin/node/resolv.conf
dockerConfig:
  dockerShimRootDirectory: /var/lib/dockershim
  dockerShimSocket: /var/run/dockershim.sock
  execHandlerName: native
enableUnidling: true
imageConfig:
  format: registry.reg-aws.openshift.com/openshift3/ose-${component}:${version}
  latest: false
iptablesSyncPeriod: 30s
kind: NodeConfig
kubeletArguments:                                                 (3)
  bootstrap-kubeconfig:
  - /etc/origin/node/bootstrap.kubeconfig
  cert-dir:
  - /etc/origin/node/certificates
  cloud-config:
  - /etc/origin/cloudprovider/aws.conf
  cloud-provider:
  - aws
  enable-controller-attach-detach:
  - 'true'
  feature-gates:
  - RotateKubeletClientCertificate=true,RotateKubeletServerCertificate=true
  node-labels:
  - node-role.kubernetes.io/compute=true
  pod-manifest-path:
  - /etc/origin/node/pods                                         (4)
  rotate-certificates:
  - 'true'
masterClientConnectionOverrides:
  acceptContentTypes: application/vnd.kubernetes.protobuf,application/json
  burst: 40
  contentType: application/vnd.kubernetes.protobuf
  qps: 20
masterKubeConfig: node.kubeconfig
networkConfig:                                                    (5)
  mtu: 8951
  networkPluginName: redhat/openshift-ovs-subnet                  (6)
servingInfo:                                                      (7)
  bindAddress: 0.0.0.0:10250
  bindNetwork: tcp4
  clientCA: client-ca.crt                                         (8)
volumeConfig:
  localQuota:
    perFSGroup: null
volumeDirectory: /var/lib/origin/openshift.local.volumes
```

(1) 認証および承認設定のオプション

(2) Pod の **/etc/resolv.conf** に追加 IP アドレスです。

3. Kubelet のコマンドライン引数 に一致する Kubelet に直接渡されるキー/値のペアです。

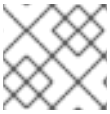
4. Pod マニフェストまたはディレクトリーへのパスです。ディレクトリーには、1 つ以上のマニフェストファイルが含まれている必要があります。OpenShift Container Platform はマニフェストファイルを使用してノードに Pod を作成します。

5. ノード上の Pod ネットワーク設定です。

6. SDN (Software defined network) プラグインです。**ovs-subnet** プラグインは **redhat/openshift-ovs-subnet**、**ovs-multitenant** プラグインは **redhat/openshift-ovs-multitenant**、または **ovs-networkpolicy** プラグインは **redhat/openshift-ovs-networkpolicy** にそれぞれ設定します。

7. ノードの証明書情報です。

8. オプション: PEM でエンコードされた証明書バンドルです。これが設定されている場合、要求ヘッダーのユーザー名をチェックする前に、有効なクライアント証明書が提示され、指定ファイルで認証局に対して検証される必要があります。



注記

/etc/origin/node/node-config.yaml ファイルは手動で変更できません。

2.11.1. ノードリソースの設定

ノードのリソースは、kubelet 引数をノード設定マップに追加して設定することができます。

1. 設定マップを編集します。

```
$ oc edit cm node-config-compute -n openshift-node
```

2. **kubeletArguments** セクションを追加して、オプションを指定します。

```
kubeletArguments:
  max-pods: 1
    - "40"
  resolv-conf: 2
    - "/etc/resolv.conf"
  image-gc-high-threshold: 3
    - "90"
  image-gc-low-threshold: 4
    - "80"
  kube-api-qps: 5
    - "20"
  kube-api-burst: 6
    - "40"
```

1. この kubelet で実行できる Pod の最大数 。

2. コンテナー DNS 解決設定のベースとして使用されるリゾルバーの設定ファイル。

3. イメージのガベージコレクションが常に実行される場合のディスク使用量のパーセント。デフォルト: 90%

4 イメージのガベージコレクションが一度も実行されない場合のディスク使用量のパーセント。ガベージコレクションの対象となる最低レベルのディスク使用量。デフォルト: 80%

5 Kubernetes API サーバーとの通信中に使用する 1 秒あたりのクエリー数 (QPS)。

6 Kubernetes API サーバーとの通信中に使用するバースト。

利用可能なすべての kubelet オプションを表示するには、以下を実行します。

```
$ hyperkube kubelet -h
```

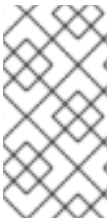
2.11.2. ノードあたりの最大 Pod 数の設定

注記

OpenShift Container Platform の各バージョンでサポートされている最大制限については、Cluster maximums ページを参照してください。

/etc/origin/node/node-config.yaml ファイルでは、1 つのパラメーターでノードにスケジュールできる Pod の最大数 **max-pods** を制御します。**max-pods** オプションを使用すると、ノード上の Pod の数が制限されます。この値を超えると、次の結果になる可能性があります。

- OpenShift Container Platform と Docker の両方で CPU 使用率が増加する。
- Pod のスケジューリングの速度が遅くなる。
- メモリー不足のシナリオが生じる可能性がある (ノードのメモリー量によって異なる)。
- IP アドレスのプールを消費する。
- リソースのオーバーコミット、およびこれによるアプリケーションのパフォーマンスの低下。



注記

Kubernetes では、単一コンテナーを保持する Pod は実際には 2 つのコンテナーを使用します。2 つ目のコンテナーは実際のコンテナーの起動前にネットワークを設定するために使用されます。そのため、10 の Pod を使用するシステムでは、実際には 20 のコンテナーが実行されていることになります。

max-pods は、ノードのプロパティーにかかわらず、ノードが実行できる Pod 数を固定値に設定します。クラスターの制限 では、**max-pods** のサポートされる最大値について説明しています。

```
kubeletArguments:
  max-pods:
    - "250"
```

上記の例を使用すると、**max-pods** のデフォルト値は **250** です。

2.12. DOCKER ストレージの再設定

コンテナーイメージをダウンロードし、コンテナーを実行して削除する際、Docker は常にマップされたディスク領域を解放する訳ではありません。結果として、一定の時間が経過するとノード上で領域不

足が生じる可能性があり、これにより OpenShift Container Platform で新規 Pod を作成できなくなるか、または Pod の作成に時間がかかる可能性があります。

たとえば、以下は 6 分が経過しても **ContainerCreating** 状態にある Pod を示しており、イベントログは FailedSync イベント を示しています。

```
$ oc get pod
```

出力例

```
NAME                            READY     STATUS              RESTARTS   AGE
cakephp-mysql-persistent-1-build   0/1       ContainerCreating   0          6m
mysql-1-9767d                      0/1       ContainerCreating   0          2m
mysql-1-deploy                     0/1       ContainerCreating   0          6m
```

```
$ oc get events
```

出力例

```
LASTSEEN   FIRSTSEEN   COUNT     NAME                            KIND                SUBOBJECT
TYPE      REASON                  SOURCE                                          MESSAGE
6m         6m          1         cakephp-mysql-persistent-1-build   Pod
Normal    Scheduled               default-scheduler                               Successfully assigned
cakephp-mysql-persistent-1-build to ip-172-31-71-195.us-east-2.compute.internal
2m         5m          4         cakephp-mysql-persistent-1-build   Pod
Warning   FailedSync              kubelet, ip-172-31-71-195.us-east-2.compute.internal   Error
syncing pod
2m         4m          4         cakephp-mysql-persistent-1-build   Pod
Normal    SandboxChanged          kubelet, ip-172-31-71-195.us-east-2.compute.internal   Pod
sandbox changed, it will be killed and re-created.
```

この問題に対する 1 つの解決法として、Docker ストレージを再設定し、Docker で不要なアーティファクトを削除することができます。

Docker ストレージを再起動するノードで、以下を実行します。

1. 以下のコマンドを実行して、ノードをスケジュール対象外としてマークします。

   ```
   $ oc adm manage-node <node> --schedulable=false
   ```

2. 以下のコマンドを実行して Docker および **atomic-openshift-node** サービスをシャットダウンします。

   ```
   $ systemctl stop docker atomic-openshift-node
   ```

3. 以下のコマンドを実行して、ローカルのボリュームディレクトリーを削除します。

   ```
   $ rm -rf /var/lib/origin/openshift.local.volumes
   ```

   このコマンドは、ローカルイメージのキャッシュをクリアします。その結果、**ose-*** イメージを含むイメージが再度プルする必要があります。これにより、イメージストアは回復しますが、Pod の起動時間が遅くなる可能性があります。

4. **/var/lib/docker** ディレクトリーを削除します。

```
$ rm -rf /var/lib/docker
```

5. 以下のコマンドを実行して Docker ストレージをリセットします。

```
$ docker-storage-setup --reset
```

6. 以下のコマンドを実行して Docker ストレージを再作成します。

```
$ docker-storage-setup
```

7. **/var/lib/docker** ディレクトリーを再作成します。

```
$ mkdir /var/lib/docker
```

8. 以下のコマンドを実行して、Docker および **atomic-openshift-node** サービスを再起動します。

```
$ systemctl start docker atomic-openshift-node
```

9. ホストを再起動してノードサービスを再起動します。

```
# systemctl restart atomic-openshift-node.service
```

10. 以下のコマンドを実行して、ノードをスケジュール対象としてマークします。

```
$ oc adm manage-node <node> --schedulable=true
```

第3章 OPENSHIFT CONTAINER PLATFORM コンポーネントの復元

3.1. 概要

OpenShift Container Platform では、クラスターおよびそのコンポーネントの 復元 は、ノードおよびアプリケーションなどの要素を別のストレージから再作成することで実行できます。

クラスターを復元するには、まず バックアップを作成 します。

重要

以下のプロセスでは、アプリケーションおよび OpenShift Container Platform クラスターを復元するための通常の方法について説明しています。ここではカスタム要件は考慮されません。クラスターを復元するために追加のアクションを実行する必要がある可能性があります。

3.2. クラスターの復元

クラスターを復元するには、まず OpenShift Container Platform を再インストールします。

手順

1. 最初に OpenShift Container Platform をインストールしたのと 同じ方法 で OpenShift Container Platform を再インストールします。

2. OpenShift Container Platform の制御下にないサービスを変更したり、モニターエージェントなどの追加サービスのインストールなど、インストール後のカスタム手順すべてを実行します。

3.3. マスターホストのバックアップの復元

重要なマスターホストファイルのバックアップを作成した後に、それらのファイルが破損するか、または間違って削除された場合は、それらのファイルをマスターにコピーし直してファイルを復元し、それらに適切なコンテンツが含まれることを確認し、影響を受けるサービスを再起動して実行できます。

手順

1. **/etc/origin/master/master-config.yaml** ファイルを復元します。

```
# MYBACKUPDIR=*/backup/$(hostname)/$(date +%Y%m%d)*
# cp /etc/origin/master/master-config.yaml /etc/origin/master/master-config.yaml.old
# cp /backup/$(hostname)/$(date +%Y%m%d)/origin/master/master-config.yaml
/etc/origin/master/master-config.yaml
# master-restart api
# master-restart controllers
```

警告

マスターサービスの再起動によりダウンタイムが生じる場合があります。ただし、マスターホストを可用性の高いロードバランサープールから削除し、復元操作を実行することができます。サービスが適切に復元された後に、マスターホストをロードバランサープールに再び追加することができます。

注記

影響を受けるインスタンスを完全に再起動して、**iptables** 設定を復元します。

2. パッケージがないために OpenShift Container Platform を再起動できない場合は、パッケージを再インストールします。

   a. 現在インストールされているパッケージの一覧を取得します。

   ```
   $ rpm -qa | sort > /tmp/current_packages.txt
   ```

   b. パッケージの一覧の間に存在する差分を表示します。

   ```
   $ diff /tmp/current_packages.txt ${MYBACKUPDIR}/packages.txt

   > ansible-2.4.0.0-5.el7.noarch
   ```

   c. 足りないパッケージを再インストールします。

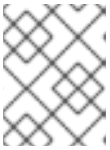
   ```
   # yum reinstall -y <packages> 1
   ```

   1 **<packages>** は、パッケージの一覧ごとに異なるパッケージに置き換えます。

3. システム証明書を **/etc/pki/ca-trust/source/anchors/** ディレクトリーにコピーして復元し、**update-ca-trust** を実行します。

```
$ MYBACKUPDIR=*/backup/$(hostname)/$(date +%Y%m%d)*
$ sudo cp ${MYBACKUPDIR}/etc/pki/ca-trust/source/anchors/<certificate> /etc/pki/ca-trust/source/anchors/ 1
$ sudo update-ca-trust
```

1 **<certificate>** は、復元するシステム証明書のファイル名に置き換えます。



注記

ファイルをコピーし直す時に、ユーザー ID およびグループ ID だけでなく、**SELinux** コンテキストも復元されていることを常に確認してください。

3.4. ノードホストバックアップの復元

重要なノードホストファイルのファイルのバックアップを作成した後に、それらのファイルが破損するか、または間違って削除された場合、これらのファイルをコピーし直してファイルを復元し、適切なコンテンツが含まれることを確認してから、影響を受けるサービスを再起動します。

手順

1. **/etc/origin/node/node-config.yaml** ファイルを復元します。

```
# MYBACKUPDIR=/backup/$(hostname)/$(date +%Y%m%d)
# cp /etc/origin/node/node-config.yaml /etc/origin/node/node-config.yaml.old
# cp /backup/$(hostname)/$(date +%Y%m%d)/etc/origin/node/node-config.yaml
/etc/origin/node/node-config.yaml
# reboot
```

警告

サービスの再起動によりダウンタイムが生じる場合があります。このプロセスを容易にするためのヒントについては、ノードの保守 を参照してください。

注記

影響を受けるインスタンスを完全に再起動して、**iptables** 設定を復元します。

1. パッケージがないために OpenShift Container Platform を再起動できない場合は、パッケージを再インストールします。
   a. 現在インストールされているパッケージの一覧を取得します。

```
$ rpm -qa | sort > /tmp/current_packages.txt
```

   b. パッケージの一覧の間に存在する差分を表示します。

```
$ diff /tmp/current_packages.txt ${MYBACKUPDIR}/packages.txt

> ansible-2.4.0.0-5.el7.noarch
```

   c. 足りないパッケージを再インストールします。

```
# yum reinstall -y <packages> 1
```

   1 **<packages>** は、パッケージの一覧ごとに異なるパッケージに置き換えます。

2. システム証明書を **/etc/pki/ca-trust/source/anchors/** ディレクトリーにコピーして復元し、**update-ca-trust** を実行します。

```
$ MYBACKUPDIR=*/backup/$(hostname)/$(date +%Y%m%d)*
$ sudo cp ${MYBACKUPDIR}/etc/pki/ca-trust/source/anchors/<certificate> /etc/pki/ca-
trust/source/anchors/
```

```
$ sudo update-ca-trust
```

<certificate> は、復元するシステム証明書のファイル名に置き換えます。

注記

ファイルをコピーし直す時に、適切なユーザー ID およびグループ ID だけでなく、**SELinux** コンテキストも復元されていることを常に確認してください。

3.5. ETCD の復元

3.5.1. etcd 設定ファイルの復元

etcd ホストが破損し、**/etc/etcd/etcd.conf** ファイルがなくなった場合は、以下の手順を使用してこれを復元します。

1. etcd ホストにアクセスします。

```
$ ssh master-0 1
```

1 **master-0** は etcd ホストの名前に置き換えます。

2. **etcd.conf** のバックアップファイルを **/etc/etcd/** にコピーします。

```
# cp /backup/etcd-config-<timestamp>/etcd/etcd.conf /etc/etcd/etcd.conf
```

3. ファイルに必要なパーミッションおよび selinux コンテキストを設定します。

```
# restorecon -RvF /etc/etcd/etcd.conf
```

この例では、バックアップファイルは、外部の NFS 共有、S3 バケットまたは他のストレージソリューションとして使用できる **/backup/etcd-config-<timestamp>/etcd/etcd.conf** パスに保存されます。

etcd 設定ファイルの復元後に、静的 Pod を再起動する必要があります。これは、etcd データの復元後に実行されます。

3.5.2. etcd データの復元

静的 Pod で etcd を復元する前に、以下を確認します。

- **etcdctl** バイナリーが利用可能であるか、またはコンテナー化インストールの場合は **rhel7/etcd** コンテナーが利用可能でなければなりません。
  以下のコマンドを実行して etcd パッケージで **etcdctl** バイナリーをインストールできます。

```
# yum install etcd
```

  このパッケージは systemd サービスもインストールします。etcd を静的 Pod で実行時に systemd サービスとして実行されないようにサービスを無効にしてマスクします。サービスを無効にしてマスクすることで、誤って開始したり、システムの再起動時に自動的にサービスの再起動がされないようにします。

```
# systemctl disable etcd.service
```

```
# systemctl mask etcd.service
```

静的 Pod で etcd を復元するには、以下を実行します。

1. Pod が実行中の場合、Pod のマニフェスト YAML ファイルを別のディレクトリーに移動して etcd Pod を停止します。

```
# mkdir -p /etc/origin/node/pods-stopped
```

```
# mv /etc/origin/node/pods/etcd.yaml /etc/origin/node/pods-stopped
```

2. 古いデータはすべて移動します。

```
# mv /var/lib/etcd /var/lib/etcd.old
```

etcdctl を使用して、Pod を復元するノードでデータを再作成します。

3. etcd スナップショットを etcd Pod のマウントパスに復元します。

```
# export ETCDCTL_API=3
```

```
# etcdctl snapshot restore /etc/etcd/backup/etcd/snapshot.db \
  --data-dir /var/lib/etcd/ \
  --name ip-172-18-3-48.ec2.internal \
  --initial-cluster "ip-172-18-3-48.ec2.internal=https://172.18.3.48:2380" \
  --initial-cluster-token "etcd-cluster-1" \
  --initial-advertise-peer-urls https://172.18.3.48:2380 \
  --skip-hash-check=true
```

バックアップの **etcd.conf** ファイルからクラスターの適切な値を取得します。

4. データディレクトリーに必要なパーミッションおよび selinux コンテキストを設定します。

```
# restorecon -RvF /var/lib/etcd/
```

5. Pod のマニフェスト YAML ファイルを必要なディレクトリーに移動して etcd Pod を再起動します。

```
# mv /etc/origin/node/pods-stopped/etcd.yaml /etc/origin/node/pods/
```

3.6. ETCD ノードの追加

etcd の復元後、他の etcd ノードをクラスターに追加できます。Ansible Playbook を使用するか、または手動の手順を実行して etcd ホストを追加できます。

3.6.1. Ansible を使用した新規 etcd ホストの追加

手順

1. Ansible インベントリーファイルで、**[new_etcd]** という名前の新規グループおよび新規ホストを作成します。次に、**new_etcd** グループを **[OSEv3]** グループの子として追加します。

```
[OSEv3:children]
masters
nodes
etcd
new_etcd (1)

... [OUTPUT ABBREVIATED] ...

[etcd]
master-0.example.com
master-1.example.com
master-2.example.com

[new_etcd] (2)
etcd0.example.com (3)
```

(1)(2)(3) これらの行を追加します。

注記

インベントリーファイル内の古い **etcd host** エントリーを新しい **etcd host** エントリーに置き換えます。古い **etcd host** を置き換えるときに、**/etc/etcd/ca/** ディレクトリーのコピーを作成する必要があります。または、**etcd hosts** をスケールアップする前に、etcd ca と証明書を再デプロイすることもできます。

2. OpenShift Container Platform をインストールし、Ansible インベントリーファイルをホストするホストから、Playbook ディレクトリーに移動し、etcd **scaleup** Playbook を実行します。

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook  playbooks/openshift-etcd/scaleup.yml
```

3. Playbook が実行された後に、新規 etcd ホストを **[new_etcd]** グループから **[etcd]** グループに移行し、現在のステータスを反映するようにインベントリーファイルを変更します。

```
[OSEv3:children]
masters
nodes
etcd
new_etcd

... [OUTPUT ABBREVIATED] ...

[etcd]
master-0.example.com
master-1.example.com
master-2.example.com
etcd0.example.com
```

4. Flannel を使用する場合には、OpenShift Container Platform のホストごとに、**/etc/sysconfig/flanneld** にある **flanneld** サービス設定を変更し、新しい etcd ホストを追加します。

```
FLANNEL_ETCD_ENDPOINTS=https://master-0.example.com:2379,https://master-
1.example.com:2379,https://master-2.example.com:2379,https://etcd0.example.com:2379
```

5. **flanneld** サービスを再起動します。

```
# systemctl restart flanneld.service
```

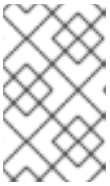
3.6.2. 新規 etcd ホストの手動による追加

etcd をマスターノードで静的 Pod として実行しない場合、別の etcd ホストを追加する必要が生じる場合があります。

手順
現在の **etcd** クラスターの変更
etcd 証明書を作成するには、値を環境の値に置き換えて **openssl** コマンドを実行します。

1. 環境変数を作成します。

```
export NEW_ETCD_HOSTNAME="*etcd0.example.com*"
export NEW_ETCD_IP="192.168.55.21"

export CN=$NEW_ETCD_HOSTNAME
export SAN="IP:${NEW_ETCD_IP}, DNS:${NEW_ETCD_HOSTNAME}"
export PREFIX="/etc/etcd/generated_certs/etcd-$CN/"
export OPENSSLCFG="/etc/etcd/ca/openssl.cnf"
```



注記

etcd_v3_ca_* として使用されるカスタムの **openssl** 拡張には、**subjectAltName** としての $SAN 環境変数が含まれます。詳細は、**/etc/etcd/ca/openssl.cnf** を参照してください。

2. 設定および証明書を保存するディレクトリーを作成します。

```
# mkdir -p ${PREFIX}
```

3. サーバー証明書要求を作成し、これに署名します (**server.csr** および **server.crt**)。

```
# openssl req -new -config ${OPENSSLCFG} \
    -keyout ${PREFIX}server.key  \
    -out ${PREFIX}server.csr \
    -reqexts etcd_v3_req -batch -nodes \
    -subj /CN=$CN

# openssl ca -name etcd_ca -config ${OPENSSLCFG} \
    -out ${PREFIX}server.crt \
    -in ${PREFIX}server.csr \
    -extensions etcd_v3_ca_server -batch
```

4. ピア証明書要求を作成し、これに署名します (**peer.csr** および **peer.crt**)。

```
# openssl req -new -config ${OPENSSLCFG} \
    -keyout ${PREFIX}peer.key \
    -out ${PREFIX}peer.csr \
    -reqexts etcd_v3_req -batch -nodes \
    -subj /CN=$CN

# openssl ca -name etcd_ca -config ${OPENSSLCFG} \
  -out ${PREFIX}peer.crt \
  -in ${PREFIX}peer.csr \
  -extensions etcd_v3_ca_peer -batch
```

5. 後で変更できるように、現在の etcd 設定および **ca.crt** ファイルをサンプルとして現在のノードからコピーします。

```
# cp /etc/etcd/etcd.conf ${PREFIX}
# cp /etc/etcd/ca.crt ${PREFIX}
```

6. 存続する etcd ホストから、新規ホストをクラスターに追加します。etcd メンバーをクラスターに追加するには、まず最初のメンバーの **peerURLs** 値のデフォルトの **localhost** ピアを調整する必要があります。

   a. **member list** コマンドを使用して最初のメンバーのメンバー ID を取得します。

```
# etcdctl --cert-file=/etc/etcd/peer.crt \
    --key-file=/etc/etcd/peer.key \
    --ca-file=/etc/etcd/ca.crt \
    --peers="https://172.18.1.18:2379,https://172.18.9.202:2379,https://172.18.0.75:2379"
\ (1)
    member list
```

   (1) **--peers** パラメーター値でアクティブな etcd メンバーのみの URL を指定するようにしてください。

   b. etcd がクラスターピアについてリッスンする IP アドレスを取得します。

```
$ ss -l4n | grep 2380
```

   c. 直前の手順で取得されたメンバー ID および IP アドレスを渡して、**etcdctl member update** コマンドを使用して **peerURLs** の値を更新します。

```
# etcdctl --cert-file=/etc/etcd/peer.crt \
    --key-file=/etc/etcd/peer.key \
    --ca-file=/etc/etcd/ca.crt \
    --peers="https://172.18.1.18:2379,https://172.18.9.202:2379,https://172.18.0.75:2379"
\
    member update 511b7fb6cc0001 https://172.18.1.18:2380
```

   d. **member list** コマンドを再実行し、ピア URL に **localhost** が含まれなくなるようにします。

7. 新規ホストを etcd クラスターに追加します。新規ホストはまだ設定されていないため、新規ホストを設定するまでステータスが **unstarted** のままであることに注意してください。

警告

各メンバーを追加し、1 回に 1 つずつメンバーをオンライン状態にします。各メンバーをクラスターに追加する際に、現在のピアの **peerURL** 一覧を調整する必要があります。**peerURL** 一覧はメンバーが追加されるたびに拡張します。**etcdctl member add** コマンドは、以下に説明されているように、メンバーを追加する際に **etcd.conf** ファイルで設定する必要のある値を出力します。

```
# etcdctl -C https://${CURRENT_ETCD_HOST}:2379 \
  --ca-file=/etc/etcd/ca.crt     \
  --cert-file=/etc/etcd/peer.crt     \
  --key-file=/etc/etcd/peer.key member add ${NEW_ETCD_HOSTNAME}
https://${NEW_ETCD_IP}:2380 1

Added member named 10.3.9.222 with ID 4e1db163a21d7651 to cluster

ETCD_NAME="<NEW_ETCD_HOSTNAME>"
ETCD_INITIAL_CLUSTER="<NEW_ETCD_HOSTNAME>=https://<NEW_HOST_IP>:2380,
<CLUSTERMEMBER1_NAME>=https:/<CLUSTERMEMBER2_IP>:2380,
<CLUSTERMEMBER2_NAME>=https:/<CLUSTERMEMBER2_IP>:2380,
<CLUSTERMEMBER3_NAME>=https:/<CLUSTERMEMBER3_IP>:2380"
ETCD_INITIAL_CLUSTER_STATE="existing"
```

1 この行で、**10.3.9.222** は etcd メンバーのラベルです。ホスト名、IP アドレスまたは単純な名前を指定できます。

8. サンプル **${PREFIX}/etcd.conf** ファイルを更新します。

   a. 以下の値を直前の手順で生成された値に置き換えます。

      - ETCD_NAME
      - ETCD_INITIAL_CLUSTER
      - ETCD_INITIAL_CLUSTER_STATE

   b. 以下の変数は、直前の手順で出力された新規ホストの IP に変更します。**${NEW_ETCD_IP}** は、値として使用できます。

```
ETCD_LISTEN_PEER_URLS
ETCD_LISTEN_CLIENT_URLS
ETCD_INITIAL_ADVERTISE_PEER_URLS
ETCD_ADVERTISE_CLIENT_URLS
```

   c. メンバーシステムを etcd ノードとして使用していた場合には、**/etc/etcd/etcd.conf** ファイルの現在の値を上書きする必要があります。

d. ファイルで構文エラーや欠落している IP アドレスがないかを確認します。 エラーや欠落がある場合には、etced サービスが失敗してしまう可能性があります。

```
# vi ${PREFIX}/etcd.conf
```

9. インストールファイルをホストするノードでは、**/etc/ansible/hosts** インベントリーファイルの **[etcd]** ホストグループを更新します。古い etcd ホストを削除し、新規ホストを追加します。

10. 証明書、サンプル設定ファイル、および **ca** を含む **tgz** ファイルを作成し、これを新規ホストにコピーします。

```
# tar -czvf /etc/etcd/generated_certs/${CN}.tgz -C ${PREFIX} .
# scp /etc/etcd/generated_certs/${CN}.tgz ${CN}:/tmp/
```

新規 **etcd** ホストの変更

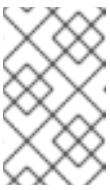
1. **iptables-services** をインストールして etcd の必要なポートを開くために iptables ユーティリティーを指定します。

```
# yum install -y iptables-services
```

2. etcd の通信を許可する **OS_FIREWALL_ALLOW** ファイアウォールルールを作成します。

- クライアントのポート 2379/tcp
- ピア通信のポート 2380/tcp

```
# systemctl enable iptables.service --now
# iptables -N OS_FIREWALL_ALLOW
# iptables -t filter -I INPUT -j OS_FIREWALL_ALLOW
# iptables -A OS_FIREWALL_ALLOW -p tcp -m state --state NEW -m tcp --dport 2379 -j
ACCEPT
# iptables -A OS_FIREWALL_ALLOW -p tcp -m state --state NEW -m tcp --dport 2380 -j
ACCEPT
# iptables-save | tee /etc/sysconfig/iptables
```



注記

この例では、新規チェーン **OS_FIREWALL_ALLOW** が作成されます。 これは、OpenShift Container Platform インストーラーがファイアウォールルールに使用する標準の名前になります。

警告

環境が IaaS 環境でホストされている場合には、インスタンスがこれらのポートに入ってくるトラフィックを許可できるように、セキュリティーグループを変更します。

3. etcd をインストールします。

```
# yum install -y etcd
```

バージョン **etcd-2.3.7-4.el7.x86_64** 以降がインストールされていることを確認します。

4. etcd Pod 定義を削除して、etcd サービスが実行されていない状態にします。

```
# mkdir -p /etc/origin/node/pods-stopped
# mv /etc/origin/node/pods/* /etc/origin/node/pods-stopped/
```

5. etcd 設定およびデータを削除します。

```
# rm -Rf /etc/etcd/*
# rm -Rf /var/lib/etcd/*
```

6. 証明書および設定ファイルを展開します。

```
# tar xzvf /tmp/etcd0.example.com.tgz -C /etc/etcd/
```

7. 新規ホストで etcd を起動します。

```
# systemctl enable etcd --now
```

8. ホストがクラスターの一部であることと現在のクラスターの正常性を確認します。

- v2 etcd api を使用する場合は、以下のコマンドを実行します。

```
# etcdctl --cert-file=/etc/etcd/peer.crt \
        --key-file=/etc/etcd/peer.key \
        --ca-file=/etc/etcd/ca.crt \
        --peers="https://*master-0.example.com*:2379,\
        https://*master-1.example.com*:2379,\
        https://*master-2.example.com*:2379,\
        https://*etcd0.example.com*:2379"\
        cluster-health
member 5ee217d19001 is healthy: got healthy result from https://192.168.55.12:2379
member 2a529ba1840722c0 is healthy: got healthy result from https://192.168.55.8:2379
member 8b8904727bf526a5 is healthy: got healthy result from
https://192.168.55.21:2379
member ed4f0efd277d7599 is healthy: got healthy result from https://192.168.55.13:2379
cluster is healthy
```

- v3 etcd api を使用する場合は、以下のコマンドを実行します。

```
# ETCDCTL_API=3 etcdctl --cert="/etc/etcd/peer.crt" \
        --key=/etc/etcd/peer.key \
        --cacert="/etc/etcd/ca.crt" \
        --endpoints="https://*master-0.example.com*:2379,\
          https://*master-1.example.com*:2379,\
          https://*master-2.example.com*:2379,\
          https://*etcd0.example.com*:2379"\
          endpoint health
https://master-0.example.com:2379 is healthy: successfully committed proposal: took =
```

```
5.011358ms
https://master-1.example.com:2379 is healthy: successfully committed proposal: took =
1.305173ms
https://master-2.example.com:2379 is healthy: successfully committed proposal: took =
1.388772ms
https://etcd0.example.com:2379 is healthy: successfully committed proposal: took =
1.498829ms
```

各 OpenShift Container Platform マスターの変更

1. すべてのマスターの **/etc/origin/master/master-config.yaml** ファイルの **etcClientInfo** セクションでマスター設定を変更します。新規 etcd ホストを、データを保存するために OpenShift Container Platform が使用する etcd サーバーの一覧に追加し、失敗したすべての etcd ホストを削除します。

```
etcdClientInfo:
  ca: master.etcd-ca.crt
  certFile: master.etcd-client.crt
  keyFile: master.etcd-client.key
  urls:
    - https://master-0.example.com:2379
    - https://master-1.example.com:2379
    - https://master-2.example.com:2379
    - https://etcd0.example.com:2379
```

2. マスター API サービスを再起動します。

   - すべてのマスターのインストールに対しては、以下を実行します。

```
# master-restart api
# master-restart controllers
```

警告

etcd ノードの数は奇数でなければなりません。 そのため、2 つ以上のホストを追加する必要があります。

3. Flannel を使用する場合、新規 etcd ホストを組み込むために、すべての OpenShift Container Platform ホストの **/etc/sysconfig/flanneld** にある **flanneld** サービス設定を変更します。

```
FLANNEL_ETCD_ENDPOINTS=https://master-0.example.com:2379,https://master-
1.example.com:2379,https://master-2.example.com:2379,https://etcd0.example.com:2379
```

4. **flanneld** サービスを再起動します。

```
# systemctl restart flanneld.service
```

3.7. OPENSHIFT CONTAINER PLATFORM サービスの再オンライン化

変更を終了した後に、OpenShift Container Platform をオンラインに戻します。

手順

1. それぞれの OpenShift Container Platform マスターで、バックアップからマスターおよびノード設定を復元し、すべての関連するサービスを有効にしてから再起動します。

```
# cp ${MYBACKUPDIR}/etc/origin/node/pods/* /etc/origin/node/pods/
# cp ${MYBACKUPDIR}/etc/origin/master/master.env /etc/origin/master/master.env
# cp ${MYBACKUPDIR}/etc/origin/master/master-config.yaml.<timestamp>
/etc/origin/master/master-config.yaml
# cp ${MYBACKUPDIR}/etc/origin/node/node-config.yaml.<timestamp>
/etc/origin/node/node-config.yaml
# cp ${MYBACKUPDIR}/etc/origin/master/scheduler.json.<timestamp>
/etc/origin/master/scheduler.json
# master-restart api
# master-restart controllers
```

2. 各 OpenShift Container Platform ノードで、必要に応じて ノードの設定マップ を更新し、**atomic-openshift-node** サービスを有効にして再起動します。

```
# cp /etc/origin/node/node-config.yaml.<timestamp> /etc/origin/node/node-config.yaml
# systemctl enable atomic-openshift-node
# systemctl start atomic-openshift-node
```

3.8. プロジェクトの復元

プロジェクトの復元には、**oc create -f <file_name>** を実行して新規プロジェクトを作成してから、エクスポートされたファイルを復元します。

手順

1. プロジェクトを作成します。

```
$ oc new-project <project_name> 1
```

1 この **<project_name>** の値は、バックアップされたプロジェクトの名前と同じである必要があります。

2. プロジェクトオブジェクトをインポートします。

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

3. ロールバインディング、シークレット、サービスアカウント、および永続ボリューム要求 (PVC) など、プロジェクトのバックアップ時にエクスポートされた他のリソースをインポートします。

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

別のオブジェクトが必要な場合に、インポートに失敗するリソースもあります。これが発生した場合は、エラーメッセージを確認し、最初にインポートする必要のあるリソースを特定します。

警告

Pod およびデフォルトサービスアカウントなどの一部のリソースは、作成できない場合があります。

3.9. アプリケーションデータの復元

rsync がコンテナーイメージ内にインストールされていることを前提とすると、アプリケーションデータは **oc rsync** コマンドを使用してバックアップできます。Red Hat **rhel7** ベースイメージには **rsync** が含まれます。したがって、**rhel7** をベースとするすべてのイメージにはこれが含まれることになります。Troubleshooting and Debugging CLI Operations - rsync を参照してください。

警告

これは、アプリケーションデータの 汎用的な 復元についての説明であり、データベースシステムの特殊なエクスポート/インポートなどのアプリケーション固有のバックアップ手順については考慮に入れられていません。

使用する永続ボリュームのタイプ (Cinder、NFS、Gluster など) によっては、他の復元手段を使用できる場合もあります。

手順

Jenkins デプロイメントのアプリケーションデータの復元例

1. バックアップを確認します。

```
$ ls -la /tmp/jenkins-backup/
total 8
drwxrwxr-x.  3 user     user   20 Sep  6 11:14 .
drwxrwxrwt. 17 root     root 4096 Sep  6 11:16 ..
drwxrwsrwx. 12 user     user 4096 Sep  6 11:14 jenkins
```

2. **oc rsync** ツールを使用して、データを実行中の Pod にコピーします。

```
$ oc rsync /tmp/jenkins-backup/jenkins jenkins-1-37nux:/var/lib
```

注記

アプリケーションによっては、アプリケーションを再起動する必要があります。

3. 必要に応じて、新規データを使ってアプリケーションを再起動します。

```
$ oc delete pod jenkins-1-37nux
```

または、デプロイメントを 0 にスケールダウンしてから再びスケールアップすることもできます。

```
$ oc scale --replicas=0 dc/jenkins
$ oc scale --replicas=1 dc/jenkins
```

3.10. PERSISTENT VOLUME CLAIM (永続ボリューム要求、PVC) の復元

このトピックでは、データを復元するための 2 つの方法について説明します。最初の方法ではファイルを削除してから、ファイルを予想される場所に戻します。2 つ目の例では、PVC を移行する方法を示します。この移行は、ストレージを移動する必要がある場合や、バックアップストレージがなくなるなどの障害発生時の状況で実行されます。

データをアプリケーションに復元する手順について、特定のアプリケーションについての復元手順を確認してください。

3.10.1. ファイルの既存 PVC への復元

手順

1. ファイルを削除します。

```
$ oc rsh demo-2-fxx6d
sh-4.2$ ls */opt/app-root/src/uploaded/*
lost+found  ocp_sop.txt
sh-4.2$ *rm -rf /opt/app-root/src/uploaded/ocp_sop.txt*
sh-4.2$ *ls /opt/app-root/src/uploaded/*
lost+found
```

2. PVC にあったファイルの rsync バックアップが含まれるサーバーのファイルを置き換えます。

```
$ oc rsync uploaded demo-2-fxx6d:/opt/app-root/src/
```

3. **oc rsh** を使用してファイルが Pod に戻されていることを確認し、Pod に接続してディレクトリーのコンテンツを表示します。

```
$ oc rsh demo-2-fxx6d
sh-4.2$ *ls /opt/app-root/src/uploaded/*
lost+found  ocp_sop.txt
```

3.10.2. データの新規 PVC への復元

以下の手順では、新規 **pvc** が作成されていることを前提としています。

手順

1. 現在定義されている **claim-name** を上書きします。

```
$ oc set volume dc/demo --add --name=persistent-volume \
  --type=persistentVolumeClaim --claim-name=filestore \ --mount-path=/opt/app-
root/src/uploaded --overwrite
```

2. Pod が新規 PVC を使用していることを確認します。

```
$ oc describe dc/demo
Name:  demo
Namespace: test
Created: 3 hours ago
Labels:  app=demo
Annotations: openshift.io/generated-by=OpenShiftNewApp
Latest Version: 3
Selector: app=demo,deployment config=demo
Replicas: 1
Triggers: Config, Image(demo@latest, auto=true)
Strategy: Rolling
Template:
  Labels: app=demo
  deployment config=demo
  Annotations: openshift.io/container.demo.image.entrypoint=["container-
entrypoint","/bin/sh","-c","$STI_SCRIPTS_PATH/usage"]
  openshift.io/generated-by=OpenShiftNewApp
  Containers:
   demo:
    Image: docker-
registry.default.svc:5000/test/demo@sha256:0a9f2487a0d95d51511e49d20dc9ff6f350436f935
968b0c83fcb98a7a8c381a
    Port: 8080/TCP
    Volume Mounts:
     /opt/app-root/src/uploaded from persistent-volume (rw)
    Environment Variables: <none>
  Volumes:
   persistent-volume:
    Type: PersistentVolumeClaim (a reference to a PersistentVolumeClaim in the same
namespace)
    *ClaimName: filestore*
    ReadOnly: false
...omitted...
```

3. デプロイメント設定では新規の **pvc** を使用しているため、**oc rsync** を実行してファイルを新規の **pvc** に配置します。

```
$ oc rsync uploaded demo-3-2b8gs:/opt/app-root/src/
sending incremental file list
uploaded/
uploaded/ocp_sop.txt
uploaded/lost+found/

sent 181 bytes  received 39 bytes  146.67 bytes/sec
total size is 32  speedup is 0.15
```

4. **oc rsh** を使用してファイルが Pod に戻されていることを確認し、Pod に接続してディレクトリーのコンテンツを表示します。

```
$ oc rsh demo-3-2b8gs
sh-4.2$ ls /opt/app-root/src/uploaded/
lost+found  ocp_sop.txt
```

第4章 マスターホストの置き換え

失敗したマスターホストを置き換えることができます。

まず、失敗したマスターホストをクラスターから削除し、次に置き換えマスターホストを追加します。失敗したマスターホストが etcd を実行していた場合、etcd を新規のマスターホストに追加して etcd を拡張します。

重要

このトピックのすべてのセクションを完了する必要があります。

4.1. マスターホストの使用の終了

マスターホストは OpenShift Container Platform API およびコントローラーサービスなどの重要なサービスを実行します。マスターホストの使用を終了するには、これらのサービスが停止している必要があります。

OpenShift Container Platform API サービスはアクティブ/アクティブサービスであるため、サービスを停止しても、要求が別のマスターサーバーに送信される限り環境に影響はありません。ただし、OpenShift Container Platform コントローラーサービスはアクティブ/パッシブサービスであり、サービスは etcd を利用してアクティブなマスターを判別します。

複数マスターアーキテクチャーでマスターホストの使用を終了するには、新しい接続でのマスターの使用を防ぐためにマスターをロードバランサープールから削除することが関係します。このプロセスは使用されるロードバランサーによって大きく異なります。以下の手順では、マスターの **haproxy** からの削除についての詳しく説明しています。OpenShift Container Platform がクラウドプロバイダーで実行されている場合や、**F5** アプライアンスを使用する場合は、特定の製品ドキュメントを参照してマスターをローテーションから削除するようにしてください。

手順

1. **/etc/haproxy/haproxy.cfg** 設定ファイルで **backend** セクションを削除します。たとえば、**haproxy** を使用して **master-0.example.com** という名前のマスターの使用を終了する場合は、ホスト名が以下から削除されていることを確認します。

```
backend mgmt8443
   balance source
   mode tcp
   # MASTERS 8443
   server master-1.example.com 192.168.55.12:8443 check
   server master-2.example.com 192.168.55.13:8443 check
```

2. 次に、**haproxy** サービスを再起動します。

```
$ sudo systemctl restart haproxy
```

3. マスターがロードバランサーから削除される場合、定義ファイルを静的 Pod のディレクトリー /etc/origin/node/pods から移動して API およびコントローラーサービスを無効にします。

```
# mkdir -p /etc/origin/node/pods/disabled
# mv /etc/origin/node/pods/controller.yaml /etc/origin/node/pods/disabled/:
+
```

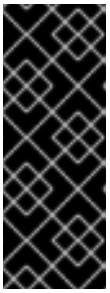
4. マスターホストはスケジュール可能な OpenShift Container Platform ノードであるため、ノードホストの使用の終了 セクションの手順に従ってください。

5. マスターホストを **/etc/ansible/hosts** Ansible インベントリーファイルの **[masters]** および **[nodes]** グループから削除し、このインベントリーファイルを使用して Ansible タスクを実行する場合の問題を回避できます。

警告

Ansible インベントリーファイルに一覧表示される最初のマスターホストの使用を終了するには、とくに注意が必要になります。

/etc/origin/master/ca.serial.txt ファイルは Ansible ホストインベントリーに一覧表示される最初のマスターでのみ生成されます。最初のマスターホストの使用を終了する場合は、このプロセスの実行前に **/etc/origin/master/ca.serial.txt** ファイルを残りのマスターホストにコピーします。



重要

複数のマスターを実行する OpenShift Container Platform 3.11 クラスターでは、マスターノードのいずれかの **/etc/origin/master**、**/etc/etcd/ca** および **/etc/etcd/generated_certs** に追加の CA 証明書が含まれます。これらは、アプリケーションノードおよび etcd ノードのスケールアップ操作に必要であり、CA ホストマスターが非推奨になっている場合は、別のマスターノードで復元する必要があります。

6. **kubernetes** サービスにはマスターホスト IP がエンドポイントとして含まれています。マスターの使用が適切に終了していることを確認するには、**kubernetes** サービスの出力を確認して、使用が終了したマスターが削除されているかどうかを確認します。

```
$ oc describe svc kubernetes -n default
Name:   kubernetes
Namespace:  default
Labels:   component=apiserver
  provider=kubernetes
Annotations:  <none>
Selector:  <none>
Type:   ClusterIP
IP:   10.111.0.1
Port:   https 443/TCP
Endpoints:  192.168.55.12:8443,192.168.55.13:8443
Port:   dns 53/UDP
Endpoints:  192.168.55.12:8053,192.168.55.13:8053
Port:   dns-tcp 53/TCP
Endpoints:  192.168.55.12:8053,192.168.55.13:8053
Session Affinity: ClientIP
Events:   <none>
```

マスターの使用が正常に終了した後、マスターが以前に実行されていたホストを安全に削除することができます。

4.2. ホストの追加

scaleup.yml Playbook を実行して新規ホストをクラスターに追加できます。この Playbook はマスターをクエリーし、新規ホストの新規証明書を生成し、配布してから、設定 Playbook を新規ホストにのみ実行します。**scaleup.yml** Playbook を実行する前に、前提条件となる ホストの準備 手順をすべて完了してください。

重要

scaleup.yml の Playbook は新規ホストの設定のみを設定します。マスターサービスの **NO_PROXY** の更新やマスターサービスの再起動は行いません。

scaleup.yml Playbook を実行するには、現在のクラスター設定を表す既存のインベントリーファイル (**/etc/ansible/hosts** など) が必要です。以前に **atomic-openshift-installer** コマンドを使用してインストールを実行した場合は、**~/.config/openshift/hosts** を調べて、インストーラーによって生成された最新のインベントリーファイルを見つけ、そのファイルをインベントリーファイルとして使用することができます。このファイルは必要に応じて変更することができます。後で **ansible-playbook** を実行する際に **-i** を使用して、そのファイルの場所を指定する必要があります。

重要

ノードの推奨の最大数については、クラスターの最大値 のセクションを参照してください。

手順

1. **openshift-ansible** パッケージを更新して最新の Playbook を取得します。

```
# yum update openshift-ansible
```

2. **/etc/ansible/hosts** ファイルを編集し、**new_<host_type>** を **[OSEv3:children]** セクションに追加します。たとえば、新規ノードホストを追加するには、**new_nodes** を追加します。

```
[OSEv3:children]
masters
nodes
new_nodes
```

新規マスターホストを追加するには、**new_masters** を追加します。

3. **[new_<host_type>]** セクションを作成して、新規ホストのホスト情報を指定します。以下の新規ノードの追加例で示されているように、既存のセクションと同じ様にこのセクションをフォーマットします。

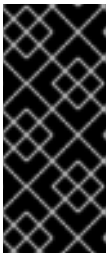
```
[nodes]
master[1:3].example.com
node1.example.com openshift_node_group_name='node-config-compute'
node2.example.com openshift_node_group_name='node-config-compute'
infra-node1.example.com openshift_node_group_name='node-config-infra'
infra-node2.example.com openshift_node_group_name='node-config-infra'
```

```
[new_nodes]
node3.example.com openshift_node_group_name='node-config-infra'
```

その他のオプションについては、ホスト変数の設定 を参照してください。

新規マスターを追加する場合は、**[new_masters]** セクションと **[new_nodes]** セクションの両方にホストを追加して、新規マスターホストが OpenShift SDN の一部となるようにします。

```
[masters]
master[1:2].example.com

[new_masters]
master3.example.com

[nodes]
master[1:2].example.com
node1.example.com openshift_node_group_name='node-config-compute'
node2.example.com openshift_node_group_name='node-config-compute'
infra-node1.example.com openshift_node_group_name='node-config-infra'
infra-node2.example.com openshift_node_group_name='node-config-infra'

[new_nodes]
master3.example.com
```



重要

マスターホストに **node-role.kubernetes.io/infra=true** ラベルを付け、それ以外に専用インフラストラクチャーノードがない場合は、エントリーに **openshift_schedulable=true** を追加してホストにスケジュール可能であることを示すマークを明示的に付ける必要もあります。そうしないと、レジストリー Pod とルーター Pod をどこにも配置できなくなります。

4. Playbook ディレクトリーに切り替え、**openshift_node_group.yml** Playbook を実行します。インベントリーファイルがデフォルトの **/etc/ansible/hosts** 以外の場所にある場合は、**-i** オプションで場所を指定します。

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook [-i /path/to/file] \
  playbooks/openshift-master/openshift_node_group.yml
```

これにより、新規ノードグループの ConfigMap が作成され、最終的にホスト上のノードの設定ファイルが作成されます。

注記

openshift_node_group.yaml Playbook を実行すると、新規ノードのみが更新されます。クラスター内の既存ノードを更新するために実行することはできません。

5. **scaleup.yml** Playbook を実行します。インベントリーファイルがデフォルトの **/etc/ansible/hosts** 以外の場所にある場合は、**-i** オプションで場所を指定します。

- ノードを追加する場合は、以下を指定します。

```
$ ansible-playbook [-i /path/to/file] \
    playbooks/openshift-node/scaleup.yml
```

- マスターを追加する場合は、以下を実行します。

```
$ ansible-playbook [-i /path/to/file] \
    playbooks/openshift-master/scaleup.yml
```

6. EFK スタックをクラスターにデプロイしている場合は、ノードラベルを **logging-infra-fluentd=true** に設定します。

```
# oc label node/new-node.example.com logging-infra-fluentd=true
```

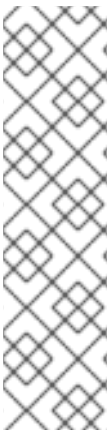
7. Playbook の実行後に、インストールの検証 を行います。

8. **[new_<host_type>]** セクションで定義したホストを適切なセクションに移動します。このようにホストを移動することで、このインベントリーファイルを使用するその後の Playbook の実行で、正しくノードが処理されるようになります。**[new_<host_type>]** セクションは空のままにできます。たとえば、新規ノードを追加する場合は、以下のように指定します。

```
[nodes]
master[1:3].example.com
node1.example.com openshift_node_group_name='node-config-compute'
node2.example.com openshift_node_group_name='node-config-compute'
node3.example.com openshift_node_group_name='node-config-compute'
infra-node1.example.com openshift_node_group_name='node-config-infra'
infra-node2.example.com openshift_node_group_name='node-config-infra'

[new_nodes]
```

4.3. ETCD のスケーリング

etcd クラスターは、リソースを etcd ホストに追加して垂直的に拡張することも、etcd ホストを追加して水平的に拡張することもできます。



注記

etcd が使用する投票システムのために、クラスターには常に奇数のメンバーが含まれている必要があります。

奇数の etcd ホストを含むクラスターの場合、フォールトトレランスに対応できます。奇数の etcd ホストがあることで、クォーラム (定足数) に必要な数が変わることはありませんが、障害発生時の耐性が高まります。たとえば、クラスターが 3 メンバーで設定される場合、クォーラム (定足数) は 2 で、1 メンバーが障害耐性用になります。これにより、クラスターはメンバーの 2 つが正常である限り、機能し続けます。

3 つの etcd ホストで設定される実稼働クラスターの使用が推奨されます。

新規ホストには、新規の Red Hat Enterprise Linux version 7 専用ホストが必要です。etcd ストレージは最大のパフォーマンスを達成できるように SSD ディスクおよび **/var/lib/etcd** でマウントされる専用ディスクに置かれる必要があります。

前提条件

1. 新規 etcd ホストを追加する前に、etcd 設定およびデータのバックアップ を行ってデータの損失を防ぎます。

2. 新規ホストが正常でないクラスターに追加されないように、現在の etcd クラスターステータスを確認します。以下のコマンドを実行します。

```
# ETCDCTL_API=3 etcdctl --cert="/etc/etcd/peer.crt" \
        --key=/etc/etcd/peer.key \
        --cacert="/etc/etcd/ca.crt" \
        --endpoints="https://*master-0.example.com*:2379,\
          https://*master-1.example.com*:2379,\
          https://*master-2.example.com*:2379"
          endpoint health
https://master-0.example.com:2379 is healthy: successfully committed proposal: took =
5.011358ms
https://master-1.example.com:2379 is healthy: successfully committed proposal: took =
1.305173ms
https://master-2.example.com:2379 is healthy: successfully committed proposal: took =
1.388772ms
```

3. **scaleup** Playbook を実行する前に、新規ホストが適切な Red Hat ソフトウェアチャンネルに登録されていることを確認します。

```
# subscription-manager register \
    --username=*<username>* --password=*<password>*
# subscription-manager attach --pool=*<poolid>*
# subscription-manager repos --disable="*"
# subscription-manager repos \
    --enable=rhel-7-server-rpms \
    --enable=rhel-7-server-extras-rpms
```

   etcd は **rhel-7-server-extras-rpms** ソフトウェアチャンネルでホストされています。

4. すべての未使用の etcd メンバーが etcd クラスターから削除されていることを確認します。これは、**scaleup** Playbook を実行する前に完了する必要があります。

   a. etcd メンバーを一覧表示します。

```
# etcdctl --cert="/etc/etcd/peer.crt" --key="/etc/etcd/peer.key" \
  --cacert="/etc/etcd/ca.crt" --endpoints=ETCD_LISTEN_CLIENT_URLS member list -w
table
```

      該当する場合は、未使用の etcd メンバー ID をコピーします。

   b. 以下のコマンドで ID を指定して、未使用のメンバーを削除します。

```
# etcdctl --cert="/etc/etcd/peer.crt" --key="/etc/etcd/peer.key" \
  --cacert="/etc/etcd/ca.crt" --endpoints=ETCD_LISTEN_CLIENT_URL member remove
UNUSED_ETCD_MEMBER_ID
```

5. 現在の etcd ノードで etcd および iptables をアップグレードします。

```
# yum update etcd iptables-services
```

6. etcd ホストの **/etc/etcd** 設定をバックアップします。

7. 新規 etcd メンバーが OpenShift Container Platform ノードでもある場合は、必要な数のホストをクラスターに追加 します。

8. この手順の残りでは 1 つのホストを追加していることを前提としていますが、複数のホストを追加する場合は、各ホストですべての手順を実行します。

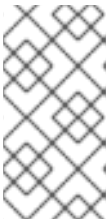
4.3.1. Ansible を使用した新規 etcd ホストの追加

手順

1. Ansible インベントリーファイルで、**[new_etcd]** という名前の新規グループおよび新規ホストを作成します。次に、**new_etcd** グループを **[OSEv3]** グループの子として追加します。

```
[OSEv3:children]
masters
nodes
etcd
new_etcd (1)

... [OUTPUT ABBREVIATED] ...

[etcd]
master-0.example.com
master-1.example.com
master-2.example.com

[new_etcd] (2)
etcd0.example.com (3)
```

(1)(2)(3)これらの行を追加します。



注記

インベントリーファイル内の古い **etcd host** エントリーを新しい **etcd host** エントリーに置き換えます。古い **etcd host** を置き換えるときに、**/etc/etcd/ca/** ディレクトリーのコピーを作成する必要があります。または、**etcd hosts** をスケールアップする前に、etcd ca と証明書を再デプロイすることもできます。

2. OpenShift Container Platform をインストールし、Ansible インベントリーファイルをホストするホストから、Playbook ディレクトリーに移動し、etcd **scaleup** Playbook を実行します。

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook  playbooks/openshift-etcd/scaleup.yml
```

3. Playbook が実行された後に、新規 etcd ホストを **[new_etcd]** グループから **[etcd]** グループに移行し、現在のステータスを反映するようにインベントリーファイルを変更します。

```
[OSEv3:children]
masters
nodes
etcd
new_etcd
```

```
... [OUTPUT ABBREVIATED] ...

[etcd]
master-0.example.com
master-1.example.com
master-2.example.com
etcd0.example.com
```

4. Flannel を使用する場合には、OpenShift Container Platform のホストごとに、**/etc/sysconfig/flanneld** にある **flanneld** サービス設定を変更し、新しい etcd ホストを追加します。

```
FLANNEL_ETCD_ENDPOINTS=https://master-0.example.com:2379,https://master-
1.example.com:2379,https://master-2.example.com:2379,https://etcd0.example.com:2379
```

5. **flanneld** サービスを再起動します。

```
# systemctl restart flanneld.service
```

4.3.2. 新規 etcd ホストの手動による追加

etcd をマスターノードで静的 Pod として実行しない場合、別の etcd ホストを追加する必要が生じる場合があります。

手順
現在の **etcd** クラスターの変更
etcd 証明書を作成するには、値を環境の値に置き換えて **openssl** コマンドを実行します。

1. 環境変数を作成します。

```
export NEW_ETCD_HOSTNAME="*etcd0.example.com*"
export NEW_ETCD_IP="192.168.55.21"

export CN=$NEW_ETCD_HOSTNAME
export SAN="IP:${NEW_ETCD_IP}, DNS:${NEW_ETCD_HOSTNAME}"
export PREFIX="/etc/etcd/generated_certs/etcd-$CN/"
export OPENSSLCFG="/etc/etcd/ca/openssl.cnf"
```

注記

etcd_v3_ca_* として使用されるカスタムの **openssl** 拡張には、**subjectAltName** としての $SAN 環境変数が含まれます。詳細は、**/etc/etcd/ca/openssl.cnf** を参照してください。

2. 設定および証明書を保存するディレクトリーを作成します。

```
# mkdir -p ${PREFIX}
```

3. サーバー証明書要求を作成し、これに署名します (**server.csr** および **server.crt**)。

```
# openssl req -new -config ${OPENSSLCFG} \
    -keyout ${PREFIX}server.key  \
    -out ${PREFIX}server.csr \
```

```
    -reqexts etcd_v3_req -batch -nodes \
    -subj /CN=$CN

# openssl ca -name etcd_ca -config ${OPENSSLCFG} \
    -out ${PREFIX}server.crt \
    -in ${PREFIX}server.csr \
    -extensions etcd_v3_ca_server -batch
```

4. ピア証明書要求を作成し、これに署名します (**peer.csr** および **peer.crt**)。

```
# openssl req -new -config ${OPENSSLCFG} \
    -keyout ${PREFIX}peer.key \
    -out ${PREFIX}peer.csr \
    -reqexts etcd_v3_req -batch -nodes \
    -subj /CN=$CN

# openssl ca -name etcd_ca -config ${OPENSSLCFG} \
  -out ${PREFIX}peer.crt \
  -in ${PREFIX}peer.csr \
  -extensions etcd_v3_ca_peer -batch
```

5. 後で変更できるように、現在の etcd 設定および **ca.crt** ファイルをサンプルとして現在のノードからコピーします。

```
# cp /etc/etcd/etcd.conf ${PREFIX}
# cp /etc/etcd/ca.crt ${PREFIX}
```

6. 存続する etcd ホストから、新規ホストをクラスターに追加します。etcd メンバーをクラスターに追加するには、まず最初のメンバーの **peerURLs** 値のデフォルトの **localhost** ピアを調整する必要があります。

   a. **member list** コマンドを使用して最初のメンバーのメンバー ID を取得します。

```
# etcdctl --cert-file=/etc/etcd/peer.crt \
    --key-file=/etc/etcd/peer.key \
    --ca-file=/etc/etcd/ca.crt \
    --peers="https://172.18.1.18:2379,https://172.18.9.202:2379,https://172.18.0.75:2379"
\ (1)
    member list
```

   (1) **--peers** パラメーター値でアクティブな etcd メンバーのみの URL を指定するようにしてください。

   b. etcd がクラスターピアについてリッスンする IP アドレスを取得します。

```
$ ss -l4n | grep 2380
```

   c. 直前の手順で取得されたメンバー ID および IP アドレスを渡して、**etcdctl member update** コマンドを使用して **peerURLs** の値を更新します。

```
# etcdctl --cert-file=/etc/etcd/peer.crt \
    --key-file=/etc/etcd/peer.key \
    --ca-file=/etc/etcd/ca.crt \
```

```
    --peers="https://172.18.1.18:2379,https://172.18.9.202:2379,https://172.18.0.75:2379"
\
    member update 511b7fb6cc0001 https://172.18.1.18:2380
```

d. **member list** コマンドを再実行し、ピア URL に **localhost** が含まれなくなるようにします。

7. 新規ホストを etcd クラスターに追加します。新規ホストはまだ設定されていないため、新規ホストを設定するまでステータスが **unstarted** のままであることに注意してください。

警告

各メンバーを追加し、1回に1つずつメンバーをオンライン状態にします。各メンバーをクラスターに追加する際に、現在のピアの **peerURL** 一覧を調整する必要があります。**peerURL** 一覧はメンバーが追加されるたびに拡張します。**etcdctl member add** コマンドは、以下に説明されているように、メンバーを追加する際に **etcd.conf** ファイルで設定する必要のある値を出力します。

```
# etcdctl -C https://${CURRENT_ETCD_HOST}:2379 \
  --ca-file=/etc/etcd/ca.crt     \
  --cert-file=/etc/etcd/peer.crt     \
  --key-file=/etc/etcd/peer.key member add ${NEW_ETCD_HOSTNAME}
https://${NEW_ETCD_IP}:2380 1

Added member named 10.3.9.222 with ID 4e1db163a21d7651 to cluster

ETCD_NAME="<NEW_ETCD_HOSTNAME>"
ETCD_INITIAL_CLUSTER="<NEW_ETCD_HOSTNAME>=https://<NEW_HOST_IP>:2380,
<CLUSTERMEMBER1_NAME>=https:/<CLUSTERMEMBER2_IP>:2380,
<CLUSTERMEMBER2_NAME>=https:/<CLUSTERMEMBER2_IP>:2380,
<CLUSTERMEMBER3_NAME>=https:/<CLUSTERMEMBER3_IP>:2380"
ETCD_INITIAL_CLUSTER_STATE="existing"
```

1 この行で、**10.3.9.222** は etcd メンバーのラベルです。ホスト名、IP アドレスまたは単純な名前を指定できます。

8. サンプル **${PREFIX}/etcd.conf** ファイルを更新します。

a. 以下の値を直前の手順で生成された値に置き換えます。

- ETCD_NAME
- ETCD_INITIAL_CLUSTER
- ETCD_INITIAL_CLUSTER_STATE

b. 以下の変数は、直前の手順で出力された新規ホストの IP に変更します。**${NEW_ETCD_IP}** は、値として使用できます。

```
ETCD_LISTEN_PEER_URLS
ETCD_LISTEN_CLIENT_URLS
ETCD_INITIAL_ADVERTISE_PEER_URLS
ETCD_ADVERTISE_CLIENT_URLS
```

c. メンバーシステムを etcd ノードとして使用していた場合には、**/etc/etcd/etcd.conf** ファイルの現在の値を上書きする必要があります。

d. ファイルで構文エラーや欠落している IP アドレスがないかを確認します。 エラーや欠落がある場合には、etced サービスが失敗してしまう可能性があります。

```
# vi ${PREFIX}/etcd.conf
```

9. インストールファイルをホストするノードでは、**/etc/ansible/hosts** インベントリーファイルの **[etcd]** ホストグループを更新します。古い etcd ホストを削除し、新規ホストを追加します。

10. 証明書、サンプル設定ファイル、および **ca** を含む **tgz** ファイルを作成し、これを新規ホストにコピーします。

```
# tar -czvf /etc/etcd/generated_certs/${CN}.tgz -C ${PREFIX} .
# scp /etc/etcd/generated_certs/${CN}.tgz ${CN}:/tmp/
```

新規 **etcd** ホストの変更

1. **iptables-services** をインストールして etcd の必要なポートを開くために iptables ユーティリティーを指定します。

```
# yum install -y iptables-services
```

2. etcd の通信を許可する **OS_FIREWALL_ALLOW** ファイアウォールルールを作成します。

- クライアントのポート 2379/tcp
- ピア通信のポート 2380/tcp

```
# systemctl enable iptables.service --now
# iptables -N OS_FIREWALL_ALLOW
# iptables -t filter -I INPUT -j OS_FIREWALL_ALLOW
# iptables -A OS_FIREWALL_ALLOW -p tcp -m state --state NEW -m tcp --dport 2379 -j
ACCEPT
# iptables -A OS_FIREWALL_ALLOW -p tcp -m state --state NEW -m tcp --dport 2380 -j
ACCEPT
# iptables-save | tee /etc/sysconfig/iptables
```

注記

この例では、新規チェーン **OS_FIREWALL_ALLOW** が作成されます。 これは、OpenShift Container Platform インストーラーがファイアウォールルールに使用する標準の名前になります。

警告

環境が IaaS 環境でホストされている場合には、インスタンスがこれらのポートに入ってくるトラフィックを許可できるように、セキュリティーグループを変更します。

3. etcd をインストールします。

```
# yum install -y etcd
```

バージョン **etcd-2.3.7-4.el7.x86_64** 以降がインストールされていることを確認します。

4. etcd Pod 定義を削除して、etcd サービスが実行されていない状態にします。

```
# mkdir -p /etc/origin/node/pods-stopped
# mv /etc/origin/node/pods/* /etc/origin/node/pods-stopped/
```

5. etcd 設定およびデータを削除します。

```
# rm -Rf /etc/etcd/*
# rm -Rf /var/lib/etcd/*
```

6. 証明書および設定ファイルを展開します。

```
# tar xzvf /tmp/etcd0.example.com.tgz -C /etc/etcd/
```

7. 新規ホストで etcd を起動します。

```
# systemctl enable etcd --now
```

8. ホストがクラスターの一部であることと現在のクラスターの正常性を確認します。

- v2 etcd api を使用する場合は、以下のコマンドを実行します。

```
# etcdctl --cert-file=/etc/etcd/peer.crt \
        --key-file=/etc/etcd/peer.key \
        --ca-file=/etc/etcd/ca.crt \
        --peers="https://*master-0.example.com*:2379,\
        https://*master-1.example.com*:2379,\
        https://*master-2.example.com*:2379,\
        https://*etcd0.example.com*:2379"\
        cluster-health
member 5ee217d19001 is healthy: got healthy result from https://192.168.55.12:2379
member 2a529ba1840722c0 is healthy: got healthy result from https://192.168.55.8:2379
member 8b8904727bf526a5 is healthy: got healthy result from
https://192.168.55.21:2379
member ed4f0efd277d7599 is healthy: got healthy result from https://192.168.55.13:2379
cluster is healthy
```

- v3 etcd api を使用する場合は、以下のコマンドを実行します。

```
# ETCDCTL_API=3 etcdctl --cert="/etc/etcd/peer.crt" \
        --key=/etc/etcd/peer.key \
        --cacert="/etc/etcd/ca.crt" \
        --endpoints="https://*master-0.example.com*:2379,\
          https://*master-1.example.com*:2379,\
          https://*master-2.example.com*:2379,\
          https://*etcd0.example.com*:2379"\
          endpoint health
https://master-0.example.com:2379 is healthy: successfully committed proposal: took =
5.011358ms
https://master-1.example.com:2379 is healthy: successfully committed proposal: took =
1.305173ms
https://master-2.example.com:2379 is healthy: successfully committed proposal: took =
1.388772ms
https://etcd0.example.com:2379 is healthy: successfully committed proposal: took =
1.498829ms
```

各 **OpenShift Container Platform** マスターの変更

1. すべてのマスターの **/etc/origin/master/master-config.yaml** ファイルの **etcClientInfo** セクションでマスター設定を変更します。新規 etcd ホストを、データを保存するために OpenShift Container Platform が使用する etcd サーバーの一覧に追加し、失敗したすべての etcd ホストを削除します。

```
etcdClientInfo:
  ca: master.etcd-ca.crt
  certFile: master.etcd-client.crt
  keyFile: master.etcd-client.key
  urls:
    - https://master-0.example.com:2379
    - https://master-1.example.com:2379
    - https://master-2.example.com:2379
    - https://etcd0.example.com:2379
```

2. マスター API サービスを再起動します。

- すべてのマスターのインストールに対しては、以下を実行します。

```
# master-restart api
# master-restart controllers
```

警告

etcd ノードの数は奇数でなければなりません。 そのため、2 つ以上のホストを追加する必要があります。

3. Flannel を使用する場合、新規 etcd ホストを組み込むために、すべての OpenShift Container Platform ホストの **/etc/sysconfig/flanneld** にある **flanneld** サービス設定を変更します。

```
FLANNEL_ETCD_ENDPOINTS=https://master-0.example.com:2379,https://master-
1.example.com:2379,https://master-2.example.com:2379,https://etcd0.example.com:2379
```

4. **flanneld** サービスを再起動します。

```
# systemctl restart flanneld.service
```

第5章 ユーザーの管理

5.1. 概要

ユーザーとは、OpenShift Container Platform API と対話するエンティティーです。ユーザーは、アプリケーションを開発する開発者の場合もあれば、クラスターを管理する管理者の場合もあります。ユーザーは、グループのすべてのメンバーに適用されるパーミッションを設定するグループに割り当てることができます。たとえば、API アクセスをグループに付与して、そのグループのすべてのメンバーに API アクセスを付与することができます。

このトピックでは、ユーザー アカウントの管理について説明します。 これには、OpenShift Container Platform で新規ユーザーアカウントを作成する方法およびそれらを削除する方法が含まれます。

5.2. ユーザーの作成

ユーザーの作成プロセスは、設定される アイデンティティープロバイダー によって異なります。デフォルトで、OpenShift Container Platform は、すべてのユーザー名およびパスワードのアクセスを拒否する **DenyAll** アイデンティティープロバイダーを使用します。

以下のプロセスでは、新規ユーザーを作成してからロールをそのユーザーに追加します。

1. アイデンティティープロバイダーに応じたユーザーアカウントを作成します。これは、アイデンティティープロバイダー設定 の一部として使用される **mappingmethod** によって異なります。

2. 新規ユーザーに必要なロールを付与します。

```
# oc create clusterrolebinding <clusterrolebinding_name> \
  --clusterrole=<role> --user=<user>
```

ここで、**--clusterrole** オプションは必要なクラスターロールになります。たとえば、新規ユーザーに対して、クラスター内のすべてに対するアクセスを付与する **cluster-admin** 権限を付与するには、以下を実行します。

```
# oc create clusterrolebinding registry-controller \
  --clusterrole=cluster-admin --user=admin
```

ロールの説明および一覧については、アーキテクチャーガイドの クラスターロールおよびローカルロール のセクションを参照してください。

クラスター管理者は、各ユーザーのアクセスレベルの管理 も実行できます。

注記

アイデンティティープロバイダーおよび定義されたグループ構造によっては、一部のロールがユーザーに自動的に付与される場合があります。詳細は、グループの LDAP との同期 についてのセクションを参照してください。

5.3. ユーザーおよび ID リストの表示

OpenShift Container Platform のユーザー設定は、OpenShift Container Platform 内の複数の場所に保存されます。アイデンティティープロバイダーの種類を問わず、OpenShift Container Platform はロールベースのアクセス制御 (RBAC) 情報およびグループメンバーシップなどの詳細情報を内部に保存しま

す。ユーザー情報を完全に削除するには、ユーザーアカウントに加えてこのデータも削除する必要があります。

OpenShift Container Platform では、2 つのオブジェクトタイプ (**user** および **identity**) に、アイデンティティープロバイダー外のユーザーデータが含まれます。

ユーザーの現在のリストを取得するには、以下を実行します。

```
$ oc get user
NAME      UID                                    FULL NAME   IDENTITIES
demo      75e4b80c-dbf1-11e5-8dc6-0e81e52cc949               htpasswd_auth:demo
```

ID の現在のリストを取得するには、以下を実行します。

```
$ oc get identity
NAME                IDP NAME        IDP USER NAME   USER NAME   USER UID
htpasswd_auth:demo  htpasswd_auth   demo            demo        75e4b80c-dbf1-11e5-8dc6-
0e81e52cc949
```

2 つのオブジェクトタイプ間で一致する UID があることに注意してください。OpenShift Container Platform の使用を開始した後に認証プロバイダーの変更を試行する場合で重複するユーザー名がある場合、そのユーザー名は、ID リストに古い認証方式を参照するエントリーがあるために機能しなくなります。

5.4. グループの作成

ユーザーは OpenShift Container Platform に要求するエンティティーである一方で、ユーザーのセットで設定される 1 つの以上のグループに編成することもできます。グループは、許可ポリシーなどの場合のように数多くのユーザーを 1 度に管理する際や、パーミッションを複数のユーザーに 1 度に付与する場合などに役立ちます。

組織が LDAP を使用している場合、LDAP レコードの OpenShift Container Platform に対する同期を実行し、複数のグループを 1 つの場所で設定できるようにすることができます。この場合、ユーザーについての情報が LDAP サーバーにあることを仮定します。詳細は、グループの LDAP との同期 についてのセクションを参照してください。

LDAP を使用していない場合は、以下の手順を使用してグループを手動で作成します。

新規グループを作成するには、以下を実行します。

```
# oc adm groups new <group_name> <user1> <user2>
```

たとえば、**west** グループを作成し、そのグループ内に **john** および **betty** ユーザーを置くには、以下を実行します。

```
# oc adm groups new west john betty
```

グループが作成されたことを確認し、グループに関連付けられたユーザーを一覧表示するには、以下を実行します。

```
# oc get groups
NAME      USERS
west      john, betty
```

次のステップ:

- ロールバインディングの管理

5.5. ユーザーおよびグループラベルの管理

ラベルをユーザーまたはグループに追加するには、以下を実行します。

```
$ oc label user/<user_name> <label_name>=<label_value>
```

たとえば、ユーザー名が **theuser** で、ラベルが **level=gold** の場合には、以下のようになります。

```
$ oc label user/theuser level=gold
```

ラベルを削除するには、以下を実行します。

```
$ oc label user/<user_name> <label_name>-
```

ユーザーまたはグループのラベルを表示するには、以下を実行します。

```
$ oc describe user/<user_name>
```

5.6. ユーザーの削除

ユーザーを削除するには、以下を実行します。

1. ユーザーレコードを削除します。

```
$ oc delete user demo
user "demo" deleted
```

2. ユーザー ID を削除します。
ユーザーの ID は使用するアイデンティティープロバイダーに関連付けられます。**oc get user** でユーザーレコードからプロバイダー名を取得します。

この例では、アイデンティティープロバイダー名は **htpasswd_auth** です。コマンドは、以下のようになります。

```
# oc delete identity htpasswd_auth:demo
identity "htpasswd_auth:demo" deleted
```

この手順を省略すると、ユーザーは再度ログインできなくなります。

上記の手順の完了後は、ユーザーが再びログインすると、新規のアカウントが OpenShift Container Platform に作成されます。

ユーザーの再ログインを防ごうとする場合 (たとえば、ある社員が会社を退職し、そのアカウントを永久に削除する必要がある場合)、そのユーザーを、設定されたアイデンティティープロバイダーの認証バックエンド (**htpasswd**、**kerberos** その他) から削除することもできます。

たとえば **htpasswd** を使用している場合、該当のユーザー名とパスワードで OpenShift Container Platform に設定された **htpasswd** ファイルのエントリーを削除します。

Lightweight Directory Access Protocol (LDAP) または Red Hat Identity Management (IdM) などの外部 ID 管理については、ユーザー管理ツールを使用してユーザーエントリーを削除します。

第6章 プロジェクトの管理

6.1. 概要

OpenShift Container Platform では、プロジェクトは関連オブジェクトを分類し、分離するために使用されます。管理者は、開発者に特定プロジェクトへのアクセスを付与し、開発者の独自プロジェクトの作成を許可したり、個別プロジェクト内の管理者権限を付与したりできます。

6.2. プロジェクトのセルフプロビジョニング

開発者の独自プロジェクトの作成を許可することができます。テンプレート に基づいてプロジェクトをプロビジョニングするエンドポイントがあります。Web コンソールおよび **oc new-project** コマンドは、開発者による 新規プロジェクトの作成 時にこのエンドポイントを使用します。

6.2.1. 新規プロジェクトのテンプレートの変更

API サーバーは、**master-config.yaml ファイル** の **projectRequestTemplate** パラメーターで識別されるテンプレートに基づいてプロジェクトを自動的にプロビジョニングします。パラメーターが定義されない場合、API サーバーは要求される名前でプロジェクトを作成するデフォルトテンプレートを作成し、要求するユーザーをプロジェクトの admin (管理者) ロールに割り当てます。

独自のカスタムプロジェクトテンプレートを作成するには、以下を実行します。

1. 現在のデフォルトプロジェクトテンプレートを使って開始します。

```
$ oc adm create-bootstrap-project-template -o yaml > template.yaml
```

2. オブジェクトを追加するか、または既存オブジェクトを変更することにより、テキストエディターを使用して **template.yaml** ファイルを変更します。

3. テンプレートを読み込みます。

```
$ oc create -f template.yaml -n default
```

4. 読み込まれたテンプレートを参照するよう **master-config.yaml** ファイルを変更します。

```
...
projectConfig:
  projectRequestTemplate: "default/project-request"
  ...
```

プロジェクト要求が送信されると、API はテンプレートで以下のパラメーターを置き換えます。

パラメーター	説明
PROJECT_NAME	プロジェクトの名前。必須。
PROJECT_DISPLAYNAME	プロジェクトの表示名。空にできます。
PROJECT_DESCRIPTION	プロジェクトの説明。空にできます。

パラメーター	説明
PROJECT_ADMIN_USER	管理ユーザーのユーザー名。
PROJECT_REQUESTING_USER	要求するユーザーのユーザー名。

API へのアクセスは、**self-provisioner** ロール と **self-provisioners** のクラスターのロールバインディングで開発者に付与されます。デフォルトで、このロールはすべての認証された開発者が利用できます。

6.2.2. セルフプロビジョニングの無効化

認証されたユーザーグループによる新規プロジェクトのセルフプロビジョニングを禁止することができます。

1. **cluster-admin** 権限を持つユーザーとしてログインします。
2. **self-provisioners** clusterrolebinding usage を確認します。以下のコマンドを実行してから、**self-provisioners** セクションの Subjects を確認します。

```
$ oc  describe clusterrolebinding.rbac self-provisioners

Name:  self-provisioners
Labels:  <none>
Annotations: rbac.authorization.kubernetes.io/autoupdate=true
Role:
  Kind: ClusterRole
  Name: self-provisioner
Subjects:
  Kind Name    Namespace
  ---- ----    ---------
  Group system:authenticated:oauth
```

3. **self-provisioner** クラスターロールをグループ **system:authenticated:oauth** から削除します。
   - **self-provisioners** クラスターロールバインディングが **self-provisioner** ロールのみを **system:authenticated:oauth** グループにバインドする場合、以下のコマンドを実行します。

     ```
     $ oc patch clusterrolebinding.rbac self-provisioners -p '{"subjects": null}'
     ```

   - **self-provisioners** クラスターロールバインディングが **self-provisioner** ロールを **system:authenticated:oauth** グループ以外のユーザー、グループまたはサービスアカウントにバインドする場合、以下のコマンドを実行します。

     ```
     $ oc adm policy remove-cluster-role-from-group self-provisioner
     system:authenticated:oauth
     ```

4. **projectRequestMessage** パラメーター値を **master-config.yaml** ファイルに設定し、開発者に対して新規プロジェクトの要求方法を指示します。このパラメーター値は、ユーザーのプロジェクトのセルフプロビジョニング試行時に web コンソールやコマンドラインに表示される文字列です。以下のメッセージのいずれかを使用できる可能性があります。

- プロジェクトを要求するには、システム管理者 (**projectname@example.com**) に問い合わせてください。
- 新規プロジェクトを要求するには、**https://internal.example.com/openshift-project-request** にあるプロジェクト要求フォームに記入します。

サンプル YAML ファイル

```
...
projectConfig:
  ProjectRequestMessage: "message"
  ...
```

5. ロールへの自動更新を防ぐには、**self-provisioners** クラスターロールバインディングを編集します。自動更新により、クラスターロールがデフォルトの状態にリセットされます。
   - ロールバインディングをコマンドラインで更新するには、以下を実行します。
     i. 次のコマンドを実行します。

     ```
     $ oc edit clusterrolebinding.rbac self-provisioners
     ```

     ii. 表示されるロールバインディングで、以下の例のように **rbac.authorization.kubernetes.io/autoupdate** パラメーター値を **false** に設定します。

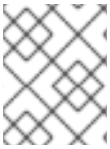
     ```
     apiVersion: authorization.openshift.io/v1
     kind: ClusterRoleBinding
     metadata:
       annotations:
         rbac.authorization.kubernetes.io/autoupdate: "false"
     ...
     ```

   - 単一コマンドを使用してロールバインディングを更新するには、以下を実行します。

     ```
     $ oc patch clusterrolebinding.rbac self-provisioners -p '{ "metadata": { "annotations": {
     "rbac.authorization.kubernetes.io/autoupdate": "false" } } }'
     ```

6.3. ノードセレクターの使用

ノードセレクターは、Pod の配置を制御するためにラベルが付けられたノードと併用されます。



注記

ラベルは、クラスターインストールの実行時 に割り当てるか、または インストール後にノードに追加 することができます。

6.3.1. クラスター全体でのデフォルトノードセレクターの設定

クラスター管理者は、クラスター全体でのノードセレクターを使用して Pod の配置を特定ノードに制限することができます。

/etc/origin/master/master-config.yaml でマスター設定ファイルを編集し、デフォルトノードセレクターの値を追加します。これは、指定された **nodeSelector** 値なしにすべてのプロジェクトで作成された Pod に適用されます。

```
...
projectConfig:
  defaultNodeSelector: "type=user-node,region=east"
...
```

変更を有効にするために OpenShift サービスを再起動します。

```
# master-restart api
# master-restart controllers
```

6.3.2. プロジェクト全体でのノードセレクターの設定

ノードセレクターを使って個々のプロジェクトを作成するには、プロジェクトの作成時に **--node-selector** オプションを使用します。たとえば、複数のリージョンを含む OpenShift Container Platform トポロジーがある場合、ノードセレクターを使用して、特定リージョンのノードにのみ Pod をデプロイするよう特定の OpenShift Container Platform プロジェクトを制限することができます。

以下では、**myproject** という名前の新規プロジェクトを作成し、Pod を **user-node** および **east** のラベルが付けられたノードにデプロイするように指定します。

```
$ oc adm new-project myproject \
    --node-selector='type=user-node,region=east'
```

このコマンドが実行されると、これが指定プロジェクト内にあるすべての Pod に対して管理者が設定するノードセレクターになります。

注記

new-project サブコマンドはクラスター管理者および開発者コマンドの **oc adm** と **oc** の両方で利用できますが、**oc adm** コマンドのみがノードセレクターを使った新規プロジェクトの作成に利用できます。**new-project** サブコマンドは、プロジェクトのセルフプロビジョニング時にプロジェクト開発者が利用することはできません。

oc adm new-project コマンドを使用すると、**annotation** セクションがプロジェクトに追加されます。プロジェクトを編集し、デフォルトを上書きするように **openshift.io/node-selector** 値を編集できます。

```
...
metadata:
  annotations:
    openshift.io/node-selector: type=user-node,region=east
...
```

また、以下のコマンドを使用して、既存プロジェクトの namespace のデフォルト値を上書きできます。

```
# oc patch namespace myproject -p \
    '{"metadata":{"annotations":{"openshift.io/node-selector":"node-role.kubernetes.io/infra=true"}}}'
```

openshift.io/node-selector が空の文字列 (**oc adm new-project --node-selector=""**) に設定される場合、プロジェクトには、クラスター全体のデフォルトが設定されている場合でも、管理者設定のノードセレクターはありません。これは、クラスター管理者はデフォルトを設定して開発者のプロジェクトをノードのサブセットに制限したり、インフラストラクチャーまたは他のプロジェクトでクラスター全体をスケジュールしたりできることを意味します。

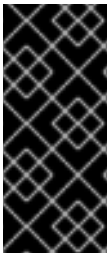
6.3.3. 開発者が指定するノードセレクター

OpenShift Container Platform 開発者は、追加でノードを制限する必要がある場合に Pod 設定でのノードセレクターの設定 を行うことができます。これはプロジェクトノードセレクターに追加されるものです。 つまり、ノードセレクターの値を持つすべてのプロジェクトについて依然としてノードセレクターの値を指定できることを意味します。

たとえば、プロジェクトが上記のアノテーションで作成 (**openshift.io/node-selector: type=user-node,region=east**) されており、開発者が別のノードセレクターをそのプロジェクトの Pod に設定する場合 (例: **clearance=classified**)、Pod はこれらの 3 つのラベル (**type=user-node**、**region=east**、および **clearance=classified**) を持つノードにのみスケジュールされます。**region=west** が Pod に設定されている場合、Pod はラベル **region=east** および **region=west** を持つノードを要求しても成功しません。ラベルは 1 つの値にのみ設定できるため、Pod はスケジュールされません。

6.4. ユーザーあたりのセルフプロビジョニングされたプロジェクト数の制限

指定されるユーザーが要求するセルフプロビジョニングされたプロジェクトの数は、**ProjectRequestLimit** 受付制御プラグイン で制限できます。



重要

OpenShift Container Platform 3.1 で、プロジェクトの要求テンプレートが 新規プロジェクトのテンプレートの変更 で説明されるプロセスを使用して作成される場合、生成されるテンプレートには、**ProjectRequestLimitConfig** に使用されるアノテーション **openshift.io/requester: ${PROJECT_REQUESTING_USER}** が含まれません。アノテーションは追加する必要があります。

ユーザーの制限を指定するには、設定をマスター設定ファイル (**/etc/origin/master/master-config.yaml**) 内のプラグインに指定する必要があります。プラグインの設定は、ユーザーラベルのセレクターの一覧および関連付けられるプロジェクト要求の最大数を取ります。

セレクターは順番に評価されます。現在のユーザーに一致する最初のセレクターは、プロジェクトの最大数を判別するために使用されます。セレクターが指定されていない場合、制限はすべてのユーザーに適用されます。プロジェクトの最大数が指定されていない場合、無制限のプロジェクトが特定のセレクターに対して許可されます。

以下の設定は、ユーザーあたりのグローバル制限を 2 プロジェクトに設定し、ラベル **level=advanced** を持つユーザーに対して 10 プロジェクトを、ラベル **level=admin** を持つユーザーに対しては無制限のプロジェクトを許可します。

```
admissionConfig:
  pluginConfig:
    ProjectRequestLimit:
      configuration:
        apiVersion: v1
        kind: ProjectRequestLimitConfig
        limits:
```

```
- selector:
    level: admin (1)
- selector:
    level: advanced (2)
  maxProjects: 10
- maxProjects: 2 (3)
```

(1) セレクター **level=admin** の場合、**maxProjects** は指定されません。これは、このラベルを持つユーザーにはプロジェクト要求の最大数が設定されないことを意味します。

(2) セレクター **level=advanced** の場合、最大数の 10 プロジェクトが許可されます。

(3) 3 つ目のエントリーにはセレクターが指定されていません。これは、セレクターが直前の 2 つのルールを満たさないユーザーに適用されることを意味します。ルールは順番に評価されるため、このルールは最後に指定する必要があります。

注記

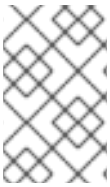
ユーザーおよびグループラベルの管理 では、ユーザーおよびグループのラベルを追加し、削除し、表示する方法について詳述しています。

変更を加えたら、OpenShift Container Platform を再起動して、変更を有効にします。

```
# master-restart api
# master-restart controllers
```

6.5. サービスアカウント別のセルフプロビジョニングされたプロジェクトの有効化および制限

デフォルトで、サービスアカウントはプロジェクトを作成できません。ただし、管理者はサービスアカウント別にこの機能を有効でき、指定されるサービスアカウントが要求するセルフプロビジョニングされたプロジェクトの数は、**ProjectRequestLimit** 受付制御プラグイン で制限できます。



注記

サービスアカウントがプロジェクトを作成することを許可される場合、プロジェクトエディターがラベルを操作する可能性があるため、プロジェクトのそれらのラベルを信頼することはできません。

1. プロジェクトにサービスアカウントを作成します (存在しない場合)。

```
$ oc create sa <sa_name>
```

2. **cluster-admin** 権限を持つユーザーとして、**self-provisioner** クラスター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 self-provisioner \
    system:serviceaccount:<project>:<sa_name>
```

3. **/etc/origin/master/master-config.yaml** のマスター設定ファイルを編集

し、**ProjectRequestLimit** セクションの **maxProjectsForServiceAccounts** パラメーター値を、任意の指定のセルフプロビジョニングされたサービスが作成できるプロジェクトの最大数に設定します。
たとえば、以下の設定は、サービスアカウントごとにグローバル制限の 3 つのプロジェクトを設定します。

```
admissionConfig:
  pluginConfig:
    ProjectRequestLimit:
      configuration:
        apiVersion: v1
        kind: ProjectRequestLimitConfig
        maxProjectsForServiceAccounts: 3
```

4. 変更を保存した後に、それらの変更を有効にするには、OpenShift Container Platform を再起動します。

```
# master-restart api
# master-restart controllers
```

5. サービスアカウントとしてログインし、新規プロジェクトを作成して、変更が適用されていることを確認します。

   a. トークンを使用し、サービスアカウントとしてログインします。

   ```
   $ oc login --token <token>
   ```

   b. 新しいプロジェクトを作成します。

   ```
   $ oc new-project <project_name>
   ```

第7章 POD の管理

7.1. 概要

このトピックでは、Pod を 1 回実行する場合の期間や使用可能な帯域幅を含む Pod の管理について説明します。

7.2. POD 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。

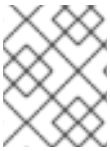
使用状況の統計を表示するには、以下を実行します。

```
$ oc adm top pods
NAME                         CPU(cores)   MEMORY(bytes)
hawkular-cassandra-1-pqx6l   219m         1240Mi
hawkular-metrics-rddnv       20m          1765Mi
heapster-n94r4               3m           37Mi
```

ラベルを持つ Pod の使用状況の統計を表示するには、以下を実行します。

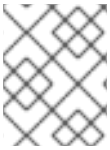
```
$ oc adm top pod --selector=''
```

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。



注記

使用状況の統計を閲覧するには、**cluster-reader** パーミッションがなければなりません。



注記

使用状況の統計を閲覧するには、**metrics-server** がインストールされている必要があります。Horizontal Pod Autoscaler の要件 を参照してください。

7.3. 1 回実行 (RUN-ONCE) POD 期間の制限

OpenShift Container Platform は 1 回実行 (run-once) Pod を使用して Pod のデプロイやビルドの実行などのタスクを実行します。1 回実行 (run-once) Pod は、**RestartPolicy** が **Never** または **OnFailure** の Pod です。

クラスター管理者は **RunOnceDuration** の受付制御プラグインを使用し、1 回実行 (run-once) Pod の有効期間の制限を強制的に実行できます。期限が切れると、クラスターはそれらの Pod をアクティブに終了しようとします。このような制限を設ける主な理由は、ビルドなどのタスクが長い時間にわたって実行されることを防ぐことにあります。

7.3.1. RunOnceDuration プラグインの設定

このプラグインの設定には、1 回実行 (run-once) Pod のデフォルト有効期限を含める必要があります。この期限はグローバルに実施されますが、プロジェクトごとに置き換えられることができます。

```
admissionConfig:
  pluginConfig:
    RunOnceDuration:
      configuration:
        apiVersion: v1
        kind: RunOnceDurationConfig
        activeDeadlineSecondsOverride: 3600 (1)
....
```

(1) 1 回実行 (run-once) Pod のグローバルのデフォルト値 (秒単位) を指定します。

7.3.2. プロジェクト別のカスタム期間の指定

1 回実行 (run-once) Pod のグローバルな最長期間を設定することに加え、管理者はアノテーション (**openshift.io/active-deadline-seconds-override**) を特定プロジェクトに追加し、グローバルのデフォルト値を上書きすることができます。

- 新規プロジェクトの場合、プロジェクト仕様の **.yaml** ファイルにアノテーションを定義します。

```
apiVersion: v1
kind: Project
metadata:
  annotations:
    openshift.io/active-deadline-seconds-override: "1000" (1)
  name: myproject
```

(1) 1 回実行 (run-once) Pod のデフォルト有効期限 (秒単位) を 1000 秒に上書きします。上書きに使用する値は、文字列形式で指定される必要があります。

- 既存プロジェクトの場合、以下を実行します。
  - **oc edit** を実行し、**openshift.io/active-deadline-seconds-override: 1000** アノテーションをエディターで追加します。

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

または

  - **oc patch** コマンドを使用します。

```
$ oc patch namespace <project_name> -p '{"metadata":{"annotations":
{"openshift.io/active-deadline-seconds-override":"1000"}}}'
```

7.3.2.1. Egress ルーター Pod のデプロイ

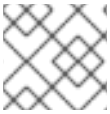
例7.1 Egress ルーターの Pod 定義の例

```
apiVersion: v1
kind: Pod
metadata:
  name: egress-1
```

```
  labels:
    name: egress-1
  annotations:
    pod.network.openshift.io/assign-macvlan: "true"
spec:
  containers:
  - name: egress-router
    image: openshift3/ose-egress-router
    securityContext:
      privileged: true
    env:
    - name: EGRESS_SOURCE 1
      value: 192.168.12.99
    - name: EGRESS_GATEWAY 2
      value: 192.168.12.1
    - name: EGRESS_DESTINATION 3
      value: 203.0.113.25
  nodeSelector:
    site: springfield-1 4
```

1. この Pod で使用するためにクラスター管理者が予約するノードサブセットの IP アドレス。
2. ノード自体で使用されるデフォルトゲートウェイと同じ値。
3. Pod の接続は 203.0.113.25 にリダイレクトされます。 ソース IP アドレスは 192.168.12.99 です。
4. Pod はラベルサイトが **springfield-1** のノードにのみデプロイされます。

pod.network.openshift.io/assign-macvlan annotation はプライマリーネットワークインターフェイスに Macvlan ネットワークインターフェイスを作成してから、それを Pod のネットワーク namespace に移行し、**egress-router** コンテナーを起動します。



注記

"true" の周りの引用符をそのまま残します。これらを省略するとエラーが生じます。

Pod には **openshift3/ose-egress-router** イメージを使用する単一コンテナーが含まれ、そのコンテナーは特権モードで実行されるので、Macvlan インターフェイスを設定したり、**iptables** ルールをセットアップしたりできます。

環境変数は **egress-router** イメージに対し、使用するアドレスを指示します。 これは、**EGRESS_SOURCE** を IP アドレスとして、また **EGRESS_GATEWAY** をゲートウェイとして使用するよう Macvlan を設定します。

NAT ルールが設定され、Pod のクラスター IP アドレスの TCP または UDP ポートへの接続が **EGRESS_DESTINATION** の同じポートにリダイレクトされるようにします。

クラスター内の一部のノードのみが指定されたソース IP アドレスを要求でき、指定されたゲートウェイを使用できる場合、受け入れ可能なノードを示す **nodeName** または **nodeSelector** を指定することができます。

7.3.2.2. Egress ルーターサービスのデプロイ

通常は、egress ルーターを参照するサービスを作成する必要があります (ただし、これは必ずしも必須ではありません)。

```
apiVersion: v1
kind: Service
metadata:
  name: egress-1
spec:
  ports:
  - name: http
    port: 80
  - name: https
    port: 443
  type: ClusterIP
  selector:
    name: egress-1
```

Pod がこのサービスに接続できるようになります。これらの接続は、予約された egress IP アドレスを使用して外部サーバーの対応するポートにリダイレクトされます。

7.3.3. Egress ファイアウォールでの Pod アクセスの制限

OpenShift Container Platform クラスター管理者は egress ポリシーを使用して、一部またはすべての Pod がクラスターからアクセスできる外部アドレスを制限できます。 これにより、以下が可能になります。

- Pod の対話を内部ホストに制限し、パブリックインターネットへの接続を開始できないようにする。
  または
- Pod の対話をパブリックインターネットに制限し、(クラスター外の) 内部ホストへの接続を開始できないようにする。
  または
- Pod が接続する理由のない指定された内部サブネット/ホストに到達できないようにする。

プロジェクトは複数の異なる egress ポリシーで設定でき、たとえば指定された IP 範囲への **<project A>** のアクセスを許可する一方で、同じアクセスを **<project B>** に対して拒否することができます。

注意

egress ポリシーで Pod のアクセスを制限するには、ovs-multitenant プラグイン を有効にする必要があります。

プロジェクト管理者は、**EgressNetworkPolicy** オブジェクトを作成することも、プロジェクトで作成するオブジェクトを編集することもできません。また、**EgressNetworkPolicy** の作成に関連して他のいくつかの制限があります。

1. デフォルト プロジェクト (および **oc adm pod-network make-projects-global** でグローバルにされたその他のプロジェクト) には egress ポリシーを設定することができません。
2. (**oc adm pod-network join-projects** を使用して) 2 つのプロジェクトをマージする場合、マージしたプロジェクトのいずれでも egress ポリシーを使用することはできません。

3. いずれのプロジェクトも複数の egress ポリシーオブジェクトを持つことができません。

上記の制限のいずれかに違反すると、プロジェクトの egress ポリシーに障害が発生し、すべての外部ネットワークトラフィックがドロップされる可能性があります。

7.3.3.1. Pod アクセス制限の設定

Pod アクセス制限を設定するには、**oc** コマンドまたは REST API を使用する必要があります。**oc [create|replace|delete]** を使用すると、**EgressNetworkPolicy** オブジェクトを操作できます。**api/swagger-spec/oapi-v1.json** ファイルには、オブジェクトを実際に機能させる方法についての API レベルの詳細情報が含まれます。

Pod のアクセス制限を設定するには、以下を実行します。

1. 対象とするプロジェクトに移動します。

2. Pod の制限ポリシーの JSON ファイルを作成します。

```
# oc create -f <policy>.json
```

3. ポリシーの詳細情報を使って JSON ファイルを設定します。以下に例を示します。

```
{
    "kind": "EgressNetworkPolicy",
    "apiVersion": "v1",
    "metadata": {
        "name": "default"
    },
    "spec": {
        "egress": [
            {
                "type": "Allow",
                "to": {
                    "cidrSelector": "1.2.3.0/24"
                }
            },
            {
                "type": "Allow",
                "to": {
                    "dnsName": "www.foo.com"
                }
            },
            {
                "type": "Deny",
                "to": {
                    "cidrSelector": "0.0.0.0/0"
                }
            }
        ]
    }
}
```

上記のサンプルがプロジェクトに追加されると、IP 範囲 **1.2.3.0/24** およびドメイン名 **www.foo.com** へのトラフィックが許可されますが、その他のすべての外部 IP アドレスへのアクセスは拒否されます。(このポリシーは外部トラフィックにのみ適用されるため、その他すべ

ての Pod へのトラフィックは影響を受けません。)

EgressNetworkPolicy のルールは順番にチェックされ、一致する最初のルールが実施されます。上記の例の 3 つの例を逆順に定義した場合、**0.0.0.0/0** ルールが最初にチェックされ、すべてのトラフィックに一致し、それらすべてを拒否するため、**1.2.3.0/24** および **www.foo.com** へのトラフィックは許可されません。

ドメイン名の更新は 30 秒以内に反映されます。上記の例で **www.foo.com** は **10.11.12.13** に解決されますが、**20.21.22.23** に変更されたとします。OpenShift Container Platform では最長 30 秒後にこれらの DNS 更新に対応します。

7.4. 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ング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。

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>
```

7.5. POD の DISRUPTION BUDGET (停止状態の予算) の設定

Pod の Disruption Budget (停止状態の予算) は、 Kubernetes API の一部であり、他の オブジェクトタイプ のように **oc** コマンドで管理できます。この設定により、メンテナーンスのためのノードのドレイン (解放) などの操作時に Pod への安全面の各種の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、ノードのメンテナーンス (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、以下の主要な部分で設定されています。

- 一連の Pod に対するラベルのクエリー機能であるラベルセレクター。
- 同期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。

以下は、**PodDisruptionBudget** リソースのサンプルです。

```
apiVersion: policy/v1beta1 (1)
kind: PodDisruptionBudget
metadata:
  name: my-pdb
spec:
  selector: (2)
    matchLabels:
      foo: bar
  minAvailable: 2 (3)
```

(1) **PodDisruptionBudget** は **policy/v1beta1** API グループの一部です。

(2) 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。

(3) 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。

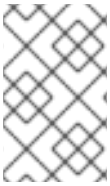
上記のオブジェクト定義で YAML ファイルを作成した場合、これを以下のようにプロジェクトに追加することができます。

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

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

NAMESPACE         NAME          MIN-AVAILABLE   SELECTOR
another-project   another-pdb   4               bar=foo
test-project      my-pdb        2               foo=bar
```

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ション の設定に基づいて、優先順位の低い Pod は Pod の Disruption Budget (停止状態の予算) の要件を無視して削除される可能性があります。

7.6. CRITICAL POD の設定

DNS など、クラスターの完全な機能に欠かせないコアコンポーネントであるものの、マスターではなく通常のクラスターノードで実行されるコアコンポーネントが多数あります。クラスターは重要なアドオンがエビクトされると正常な機能を停止する可能性があります。**scheduler.alpha.kubernetes.io/critical-pod** アノテーションを Pod 仕様に追加し、descheduler がこれらの Pod を削除できないようにすることで、Pod を Critical Pod にすることができます。

```
spec:
  template:
    metadata:
      name: critical-pod
      annotations:
        scheduler.alpha.kubernetes.io/critical-pod: "true"
```

第8章 ネットワークの管理

8.1. 概要

このトピックでは、プロジェクトの分離および発信トラフィックの制御を含む、全般的な クラスターネットワーク の管理について説明します。

Pod ごとの帯域幅の制限などの Pod レベルのネットワーク機能については、Pod の管理 で説明されています。

8.2. POD ネットワークの管理

クラスターが **ovs-multitenant** SDN プラグイン を使用するように設定されている場合、管理者 CLI を使用してプロジェクトの別個の Pod オーバーレイネットワークを管理することができます。必要な場合は、プラグイン設定手順について Configuring the SDN セクションを参照してください。

8.2.1. プロジェクトネットワークへの参加

プロジェクトを既存のプロジェクトネットワークに参加させるには、以下を実行します。

```
$ oc adm pod-network join-projects --to=<project1> <project2> <project3>
```

上記の例で、**<project2>** および **<project3>** のすべての Pod およびサービスから、**<project1>** のすべての Pod およびサービスへのアクセスが可能となり、その逆の場合も可能になります。サービスは、IP または完全修飾 DNS 名 (**<service>.<pod_namespace>.svc.cluster.local**) のいずれかでアクセスできます。たとえば、プロジェクト **myproject** の **db** という名前のサービスにアクセスするには、**db.myproject.svc.cluster.local** を使用します。

または、特定のプロジェクト名を指定する代わりに **--selector=<project_selector>** オプションを使用することもできます。

組み合わせたネットワークを確認するには、以下を実行します。

```
$ oc get netnamespaces
```

次に **NETID** 列を確認します。同じ Pod ネットワークのプロジェクトには同じ NetID があります。

8.3. プロジェクトネットワークの分離

プロジェクトネットワークをクラスターから分離したり、その逆を実行するには、以下を実行します。

```
$ oc adm pod-network isolate-projects <project1> <project2>
```

上記の例では、**<project1>** および **<project2>** のすべての Pod およびサービスは、クラスター内のグローバル以外のプロジェクトの Pod およびサービスに アクセスできず、その逆も実行できません。

または、特定のプロジェクト名を指定する代わりに **--selector=<project_selector>** オプションを使用することもできます。

8.3.1. プロジェクトネットワークのグローバル化

プロジェクトからクラスター内のすべての Pod およびサービスにアクセスできるようにするか、その逆を可能にするには、以下を実行します。

```
$ oc adm pod-network make-projects-global <project1> <project2>
```

上記の例では、**<project1>** および **<project2>** のすべての Pod およびサービスは、クラスター内のすべての Pod およびサービスにアクセスできるようになり、その逆の場合も可能になります。

または、特定のプロジェクト名を指定する代わりに **--selector=<project_selector>** オプションを使用することもできます。

8.4. ルートおよび INGRESS オブジェクトにおけるホスト名の競合防止の無効化

OpenShift Container Platform では、ルートおよび ingress オブジェクトのホスト名の競合防止はデフォルトで有効にされています。これは、**cluster-admin** ロールのないユーザーは、作成時にのみルーターまたは ingress オブジェクトのホスト名を設定でき、その後は変更できなくなることを意味しています。ただし、ルートおよび ingress オブジェクトのこの制限は、一部またはすべてのユーザーに対して緩和することができます。

警告

OpenShift Container Platform はオブジェクト作成のタイムスタンプを使用して特定のホスト名の最も古いルートや ingress オブジェクトを判別するため、ルートまたは ingress オブジェクトは、古いルートがそのホスト名を変更したり、ingress オブジェクトが導入される場合に新規ルートのホスト名をハイジャックする可能性があります。

OpenShift Container Platform クラスター管理者は、作成後でもルートのホスト名を編集できます。また、特定のユーザーがこれを実行できるようにロールを作成することもできます。

```
$ oc create clusterrole route-editor --verb=update --resource=routes.route.openshift.io/custom-host
```

次に、新規ロールをユーザーにバインドできます。

```
$ oc adm policy add-cluster-role-to-user route-editor user
```

ingress オブジェクトのホスト名の競合防止を無効にすることもできます。これを実行することで、**cluster-admin** ロールを持たないユーザーが作成後も ingress オブジェクトのホスト名を編集できるようになります。これは、ingress オブジェクトのホスト名の編集を許可する場合などに Kubernetes の動作に依存する OpenShift Container Platform のインストールで役に立ちます。

1. 以下を **master.yaml** ファイルに追加します。

```
admissionConfig:
  pluginConfig:
    openshift.io/IngressAdmission:
      configuration:
        apiVersion: v1
```

```
    allowHostnameChanges: true
    kind: IngressAdmissionConfig
  location: ""
```

2. 変更を有効にするために、マスターサービスを再起動します。

```
$ master-restart api
$ master-restart controllers
```

8.5. EGRESS トラフィックの制御

クラスター管理者は、ホストレベルで数多くの静的 IP アドレスを特定ノードに割り当てることができます。アプリケーション開発者がそれぞれのアプリケーションサービスに専用 IP アドレスを必要とする場合、ファイアウォールアクセスを要求するプロセスでこのアドレスを要求することができます。その後、開発者はデプロイメント設定の **nodeSelector** を使用して、開発者のプロジェクトから egress ルーターをデプロイし、静的 IP アドレスが事前に割り当てられたホストに Pod が到達することを確認できます。

egress Pod のデプロイメントでは、宛先に到達するために必要なソース IP のいずれか、保護されるサービスの宛先 IP、およびゲートウェイ IP を宣言します。Pod のデプロイ後は、サービスを作成 して、egress ルーター Pod にアクセスし、そのソース IP を企業ファイアウォールに追加できます。その後、開発者はプロジェクトで作成された egress ルーターサービスへのアクセス情報を取得します (例: **service.project.cluster.domainname.com**)。

開発者が外部の firewalled サービスにアクセスする必要がある場合、実際の保護されたサービス URL ではなくアプリケーション (例: JDBC 接続情報) で、egress ルーター Pod のサービス (**service.project.cluster.domainname.com**) に対して呼び出し実行することができます。

さらに、静的 IP アドレスをプロジェクトに割り当て、指定されたプロジェクトからの発信外部接続すべてに認識可能な起点を設定できます。これは、トラフィックと特定の宛先に送信するために使用されるデフォルトの egress ルーターとは異なります。

詳細は、外部プロジェクトトラフィックの固定 IP の有効化 セクションを参照してください。

OpenShift Container Platform クラスター管理者は、以下を使用して egress トラフィックを制御できます。

ファイアウォール

: egress ファイアウォールを使用すると、受け入れ可能な発信トラフィックポリシーを実施し、特定のエンドポイントまたは IP 範囲 (サブネット) のみを動的エンドポイント (OpenShift Container Platform 内の Pod) が通信できる受け入れ可能なターゲットとすることができます。

ルーター

: egress ルーターを使用することで、識別可能なサービスを作成し、トラフィックを特定の宛先に送信できます。これにより、それらの外部の宛先はトラフィックを既知のソースから送られるものとして処理します。 これにより namespace の特定の Pod のみがトラフィックをデータベースにプロキシー送信するサービス (egress ルーター) と通信できるよう外部データベースが保護されるため、セキュリティー対策として役立ちます。

iptables

: 上記の OpenShift Container Platform 内のソリューションのほかにも、発信トラフィックに適用される iptables ルールを作成することができます。これらのルールは、egress ファイアウォールよりも多くのオプションを許可しますが、特定のプロジェクトに制限することはできません。

8.6. 外部リソースへのアクセスを制限するための EGRESS ファイアウォールの使用

OpenShift Container Platform クラスター管理者は egress ファイアウォールを使用して、一部またはすべての Pod がクラスター内からアクセスできる外部 IP アドレスを制限できます。egress ファイアウォールポリシーは以下のシナリオをサポートします。

- Pod の接続を内部ホストに制限し、パブリックインターネットへの接続を開始できないようにする。
- Pod の接続をパブリックインターネットに制限し、OpenShift Container Platform クラスター外にある内部ホストへの接続を開始できないようにする。
- Pod が到達不能な状態の指定された内部サブネットまたはホストに到達できないようにする。

egress ポリシーは、IP アドレス範囲を CIDR 形式で指定するか、または DNS 名を指定して設定できます。たとえば、指定された IP 範囲への **<project_A>** アクセスを許可しつつ、**<project_B>** への同じアクセスを拒否することができます。または、アプリケーション開発者が (Python) pip mirror からの更新を制限したり、更新を承認されたソースからの更新のみに強制的に制限したりすることができます。

注意

Pod アクセスを egress ポリシーで制限するには、**ovs-multitenant** または **ovs-networkpolicy** プラグインを有効にする 必要があります。

ovs-multitenant プラグインを使用している場合、egress ポリシーはプロジェクトごとに 1 つのポリシーとのみ互換性を持ち、グローバルプロジェクトなどのネットワークを共有するプロジェクトでは機能しません。

プロジェクト管理者は、**EgressNetworkPolicy** オブジェクトを作成することも、プロジェクトで作成するオブジェクトを編集することもできません。また、**EgressNetworkPolicy** の作成に関連して他のいくつかの制限があります。

- デフォルト プロジェクト (および **oc adm pod-network make-projects-global** でグローバルにされたその他のプロジェクト) には egress ポリシーを設定することができません。
- (**oc adm pod-network join-projects** を使用して) 2 つのプロジェクトをマージする場合、マージしたプロジェクトのいずれでも egress ポリシーを使用することはできません。
- いずれのプロジェクトも複数の egress ポリシーオブジェクトを持つことができません。

上記の制限のいずれかに違反すると、プロジェクトの egress ポリシーに障害が発生し、すべての外部ネットワークトラフィックがドロップされる可能性があります。

oc コマンドまたは REST API を使用して egress ポリシーを設定します。 **oc [create|replace|delete]** を使用すると、**EgressNetworkPolicy** オブジェクトを操作できます。**api/swagger-spec/oapi-v1.json** ファイルには、オブジェクトを実際に機能させる方法についての API レベルの詳細情報が含まれます。

egress ポリシーを設定するには、以下を実行します。

1. 対象とするプロジェクトに移動します。
2. 以下の例のように、使用する必要のあるポリシー設定で JSON ファイルを作成します。

```
{
    "kind": "EgressNetworkPolicy",
```

```
    "apiVersion": "v1",
    "metadata": {
        "name": "default"
    },
    "spec": {
        "egress": [
            {
                "type": "Allow",
                "to": {
                    "cidrSelector": "1.2.3.0/24"
                }
            },
            {
                "type": "Allow",
                "to": {
                    "dnsName": "www.foo.com"
                }
            },
            {
                "type": "Deny",
                "to": {
                    "cidrSelector": "0.0.0.0/0"
                }
            }
        ]
    }
}
```

上記のサンプルがプロジェクトに追加されると、IP 範囲 **1.2.3.0/24** およびドメイン名 **www.foo.com** へのトラフィックが許可されますが、その他のすべての外部 IP アドレスへのアクセスは拒否されます。このポリシーは外部トラフィックにのみ適用されるため、その他すべての Pod へのトラフィックは影響を受けません。

EgressNetworkPolicy のルールは順番にチェックされ、一致する最初のルールが実施されます。上記の例の 3 つの例を逆順に定義した場合、**0.0.0.0/0** ルールが最初にチェックされ、すべてのトラフィックに一致し、それらすべてを拒否するため、**1.2.3.0/24** および **www.foo.com** へのトラフィックは許可されません。

ドメイン名の更新は、ローカルの非権威サーバーのドメインの TTL (time to live) 値に基づいてポーリングされます。Pod は必要な場合には、同じローカルのネームサーバーのドメインを解決する必要もあります。そうしないと、egress ネットワークポリシーコントローラーと Pod で認識されるドメインの IP アドレスが異なり、egress ネットワークが予想通りに実施されない場合があります。egress ネットワークポリシーコントローラーおよび Pod は同じローカルネームサーバーを非同期にポーリングするため、Pod が egress コントローラーの前に更新された IP を取得するという競合状態が発生する可能性があります。この現時点の制限により、**EgressNetworkPolicy** のドメイン名の使用は、IP アドレスの変更が頻繁に生じないドメインの場合にのみ推奨されます。

注記

egress ファイアウォールは、DNS 解決用に Pod が置かれるノードの外部インターフェイスに Pod が常にアクセスできるようにします。DNS 解決がローカルノード上のいずれかによって処理されない場合は、Pod でドメイン名を使用している場合には DNS サーバーの IP アドレスへのアクセスを許可する egress ファイアウォールを追加する必要があります。

3. JSON ファイルを使用して EgressNetworkPolicy オブジェクトを作成します。

```
$ oc create -f <policy>.json
```

注意

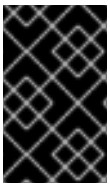
ルート を作成してサービスを公開すると、**EgressNetworkPolicy** は無視されます。Egress ネットワークポリシーサービスのエンドポイントのフィルターは、ノード **kubeproxy** で実行されます。ルーターが使用される場合は、**kubeproxy** はバイパスされ、egress ネットワークポリシーの施行は適用されません。管理者は、ルートを作成するためのアクセスを制限してこのバイパスを防ぐことができます。

8.6.1. 外部リソースから Pod トラフィックを認識可能にするための Egress ルーターの使用

OpenShift Container Platform egress ルーターは、他の用途で使用されていないプライベートソース IP アドレスを使用して、指定されたリモートサーバーにトラフィックをリダイレクトするサービスを実行します。このサービスにより、Pod はホワイトリスト IP アドレスからのアクセスのみを許可するように設定されたサーバーと通信できるようになります。

重要

egress ルーターはすべての発信接続のために使用されることが意図されていません。多数の egress ルーターを作成することで、ネットワークハードウェアの制限を引き上げる可能性があります。たとえば、すべてのプロジェクトまたはアプリケーションに egress ルーターを作成すると、ソフトウェアの MAC アドレスのフィルターにフォールバックする前にネットワークインターフェイスが処理できるローカル MAC アドレス数の上限を超えてしまう可能性があります。



重要

現時点で、egress ルーターには Amazon AWS, Azure Cloud またはレイヤー 2 操作をサポートしないその他のクラウドプラットフォームとの互換性がありません。 それらに macvlan トラフィックとの互換性がないためです。

デプロイメントに関する考慮事項

Egress ルーターは 2 つ目の IP アドレスおよび MAC アドレスをノードのプライマリーネットワークインターフェイスに追加します。OpenShift Container Platform をベアメタルで実行していない場合は、ハイパーバイザーまたはクラウドプロバイダーが追加のアドレスを許可するように設定する必要があります。

Red Hat OpenStack Platform

OpenShift Container Platform を Red Hat OpenStack Platform にデプロイしている場合、OpenStack 環境で IP および MAC アドレスのホワイトリストを作成する必要があります。作成しないと、通信は失敗します。

```
neutron port-update $neutron_port_uuid \
  --allowed_address_pairs list=true \
  type=dict mac_address=<mac_address>,ip_address=<ip_address>
```

Red Hat Enterprise Virtualization

Red Hat Enterprise Virtualization を使用している場合は、**EnableMACAntiSpoofingFilterRules** を **false** に設定する必要が あります。

VMware vSphere

VMware vSphere を使用している場合は、vSphere 標準スイッチのセキュリティー保護についての VMWare ドキュメント を参照してください。vSphere Web クライアントからホストの仮想スイッチを選択して、VMWare vSphere デフォルト設定を表示し、変更します。

とくに、以下が有効にされていることを確認します。

- MAC アドレスの変更
- 偽装転送 (Forged Transit)
- 無作為別モード (Promiscuous Mode) 操作

Egress ルーターモード

egress ルーターは、リダイレクトモード と HTTP プロキシーモード および DNS プロキシーモード の 3 つの異なるモードで実行できます。リダイレクトモードは、HTTP および HTTPS 以外のすべてのサービスで機能します。HTTP および HTTPS サービスの場合は、HTTP プロキシーモードを使用します。IP アドレスまたはドメイン名を持つ TCP ベースのサービスの場合は、DNS プロキシーモードを使用します。

8.6.1.1. リダイレクトモードでの Egress ルーター Pod のデプロイ

リダイレクトモードでは、egress ルーターは、トラフィックを独自の IP アドレスから 1 つ以上の宛先 IP アドレスにリダイレクトするために iptables ルールをセットアップします。予約されたソース IP アドレスを使用する必要のあるクライアント Pod は、宛先 IP に直接接続するのでなく、egress ルーターに接続するように変更される必要があります。

1. 以下を使用して Pod 設定を作成します。

```
apiVersion: v1
kind: Pod
metadata:
  name: egress-1
  labels:
    name: egress-1
  annotations:
    pod.network.openshift.io/assign-macvlan: "true" (1)
spec:
  initContainers:
  - name: egress-router
    image: registry.redhat.io/openshift3/ose-egress-router
    securityContext:
      privileged: true
    env:
    - name: EGRESS_SOURCE (2)
      value: 192.168.12.99/24
    - name: EGRESS_GATEWAY (3)
      value: 192.168.12.1
    - name: EGRESS_DESTINATION (4)
      value: 203.0.113.25
    - name: EGRESS_ROUTER_MODE (5)
      value: init
```

```
containers:
- name: egress-router-wait
  image: registry.redhat.io/openshift3/ose-pod
nodeSelector:
  site: springfield-1 6
```

1. プライマリーネットワークインターフェイスで Macvlan ネットワークインターフェイスを作成し、これを Pod のネットワークプロジェクトに移行してから **egress-router** コンテナーを起動します。**"true"** の周りの引用符をそのまま残します。これらを省略すると、エラーが発生します。プライマリーネットワークインターフェイス以外のネットワークインターフェイスで Macvlan インターフェイスを作成するには、アノテーションの値を該当インターフェイスの名前に設定します。たとえば、 **eth1** を使用します。
2. ノードが置かれており、クラスター管理者がこの Pod で使用するために予約している物理ネットワークの IP アドレスです。オプションとして、サブネットの長さ **/24** 接尾辞を組み込み、ローカルサブネットへの適切なルートがセットアップされるようにできます。サブネットの長さを指定しない場合、egress ルーターは **EGRESS_GATEWAY** 変数で指定されたホストにのみアクセスでき、サブネットの他のホストにはアクセスできません。
3. ノードで使用されるデフォルトゲートウェイと同じ値です。
4. トラフィックの送信先となる外部サーバー。この例では、Pod の接続は 203.0.113.25 にリダイレクトされます。 ソース IP アドレスは 192.168.12.99 です。
5. これは egress ルーターイメージに対して、これが init コンテナーとしてデプロイされていることを示しています。以前のバージョンの OpenShift Container Platform (および egress ルーターイメージ) はこのモードをサポートしておらず、通常のコンテナーとして実行される必要がありました。
6. Pod はラベル **site=springfield-1** の設定されたノードにのみデプロイされます。

2. 上記の定義を使用して Pod を作成します。

```
$ oc create -f <pod_name>.json
```

Pod が作成されているかどうかを確認するには、以下を実行します。

```
$ oc get pod <pod_name>
```

3. egresss ルーターを参照するサービスを作成し、他の Pod が Pod の IP アドレスを見つけられるようにします。

```
apiVersion: v1
kind: Service
metadata:
  name: egress-1
spec:
  ports:
  - name: http
    port: 80
  - name: https
    port: 443
```

```
  type: ClusterIP
  selector:
    name: egress-1
```

Pod がこのサービスに接続できるようになります。これらの接続は、予約された egress IP アドレスを使用して外部サーバーの対応するポートにリダイレクトされます。

egress ルーターのセットアップは、**openshift3/ose-egress-router** イメージで作成される init コンテナーで実行され、このコンテナーは Macvlan インターフェイスを設定し、**iptables** ルールをセットアップできるように特権モード実行されます。**iptables** ルールのセットアップ終了後に、これは終了し、**openshift3/ose-pod** コンテナーが Pod が強制終了されるまで (特定のタスクを実行しない) 実行状態になります。

環境変数は **egress-router** イメージに対し、使用するアドレスを指示します。 これは、**EGRESS_SOURCE** を IP アドレスとして、また **EGRESS_GATEWAY** をゲートウェイとして使用するよう Macvlan を設定します。

NAT ルールが設定され、Pod のクラスター IP アドレスの TCP または UDP ポートへの接続が **EGRESS_DESTINATION** の同じポートにリダイレクトされるようにします。

クラスター内の一部のノードのみが指定されたソース IP アドレスを要求でき、指定されたゲートウェイを使用できる場合、受け入れ可能なノードを示す **nodeName** または **nodeSelector** を指定することができます。

8.6.1.2. 複数の宛先へのリダイレクト

前の例では、任意のポートでの egress Pod (またはその対応するサービス) への接続は単一の宛先 IP にリダイレクトされます。ポートに応じて異なる宛先 IP を設定することもできます。

```
apiVersion: v1
kind: Pod
metadata:
  name: egress-multi
  labels:
    name: egress-multi
  annotations:
    pod.network.openshift.io/assign-macvlan: "true"
spec:
  initContainers:
  - name: egress-router
    image: registry.redhat.io/openshift3/ose-egress-router
    securityContext:
      privileged: true
    env:
    - name: EGRESS_SOURCE (1)
      value: 192.168.12.99/24
    - name: EGRESS_GATEWAY
      value: 192.168.12.1
    - name: EGRESS_DESTINATION (2)
      value: |
        80   tcp 203.0.113.25
        8080 tcp 203.0.113.26 80
        8443 tcp 203.0.113.26 443
        203.0.113.27
    - name: EGRESS_ROUTER_MODE
```

```
    value: init
containers:
- name: egress-router-wait
  image: registry.redhat.io/openshift3/ose-pod
```

1 ノードが置かれており、クラスター管理者がこの Pod で使用するために予約している物理ネットワークの IP アドレスです。オプションとして、サブネットの長さ **/24** 接尾辞を組み込み、ローカルサブネットへの適切なルートがセットアップされるようにできます。サブネットの長さを指定しない場合、egress ルーターは **EGRESS_GATEWAY** 変数で指定されたホストにのみアクセスでき、サブネットの他のホストにはアクセスできません。

2 **EGRESS_DESTINATION** はその値に YAML 構文を使用し、複数行の文字列を使用できます。詳細は、以下を参照してください。

EGRESS_DESTINATION の各行は、以下の 3 つのタイプのいずれかになります。

- **<port> <protocol> <IP_address>**: これは、指定される **<port>** への着信接続が指定される **<IP_address>** の同じポートにリダイレクトされる必要があることを示しています。**<protocol>** は **tcp** または **udp** のいずれかになります。上記の例では、最初の行がローカルポート 80 から 203.0.113.25 のポート 80 にトラフィックをリダイレクトしています。

- **<port> <protocol> <IP_address> <remote_port>**: 接続が **<IP_address>** の別の **<remote_port>** にリダイレクトされるのを除き、上記と同じになります。この例では、2 番目と 3 番目の行ではローカルポート 8080 および 8443 を 203.0.113.26 のリモートポート 80 および 443 にリダイレクトしています。

- **<fallback_IP_address>**: **EGRESS_DESTINATION** の最後の行が単一 IP アドレスである場合、それ以外のポートの接続はその IP アドレス (上記の例では 203.0.113.27) の対応するポートにリダイレクトされます。フォールバック IP アドレスがない場合、他のポートでの接続は単純に拒否されます。)

8.6.1.3. ConfigMap の使用による EGRESS_DESTINATION の指定

宛先マッピングのセットのサイズが大きいか、またはこれが頻繁に変更される場合、ConfigMap を使用して一覧を外部で維持し、egress ルーター Pod がそこから一覧を読み取れるようにすることができます。これには、プロジェクト管理者が ConfigMap を編集できるという利点がありますが、これには特権付きコンテナーが含まれるため、管理者は Pod 定義を直接編集することはできません。

1. **EGRESS_DESTINATION** データを含むファイルを作成します。

```
$ cat my-egress-destination.txt
# Egress routes for Project "Test", version 3

80   tcp 203.0.113.25

8080 tcp 203.0.113.26 80
8443 tcp 203.0.113.26 443

# Fallback
203.0.113.27
```

   空の行とコメントをこのファイルに追加できることに注意してください。

2. このファイルから ConfigMap オブジェクトを作成します。

```
$ oc delete configmap egress-routes --ignore-not-found
$ oc create configmap egress-routes \
  --from-file=destination=my-egress-destination.txt
```

ここで、**egress-routes** は作成される ConfigMap オブジェクトの名前で、**my-egress-destination.txt** はデータの読み取り元のファイルの名前です。

3. 前述のように egress ルーター Pod 定義を作成しますが、ConfigMap を環境セクションの **EGRESS_DESTINATION** に指定します。

```
...
env:
- name: EGRESS_SOURCE 1
  value: 192.168.12.99/24
- name: EGRESS_GATEWAY
  value: 192.168.12.1
- name: EGRESS_DESTINATION
  valueFrom:
    configMapKeyRef:
      name: egress-routes
      key: destination
- name: EGRESS_ROUTER_MODE
  value: init
...
```

1 ノードが置かれており、クラスター管理者がこの Pod で使用するために予約している物理ネットワークの IP アドレスです。オプションとして、サブネットの長さ **/24** 接尾辞を組み込み、ローカルサブネットへの適切なルートがセットアップされるようにできます。サブネットの長さを指定しない場合、egress ルーターは **EGRESS_GATEWAY** 変数で指定されたホストにのみアクセスでき、サブネットの他のホストにはアクセスできません。

注記

egress ルーターは、ConfigMap が変更されても自動的に更新されません。更新を取得するには Pod を再起動します。

8.6.1.4. Egress ルーター HTTP プロキシー Pod のデプロイ

HTTP プロキシーモードでは、egress ルーターはポート **8080** で HTTP プロキシーとして実行されます。これは、HTTP または HTTPS ベースのサービスと通信するクライアントの場合にのみ機能しますが、通常それらを機能させるのにクライアント Pod への多くの変更は不要です。環境変数を設定することで、プログラムは HTTP プロキシーを使用するように指示されます。

1. 例として以下を使用して Pod を作成します。

```
apiVersion: v1
kind: Pod
metadata:
  name: egress-http-proxy
  labels:
    name: egress-http-proxy
  annotations:
    pod.network.openshift.io/assign-macvlan: "true" 1
spec:
```

```
initContainers:
- name: egress-router-setup
  image: registry.redhat.io/openshift3/ose-egress-router
  securityContext:
    privileged: true
  env:
  - name: EGRESS_SOURCE (2)
    value: 192.168.12.99/24
  - name: EGRESS_GATEWAY (3)
    value: 192.168.12.1
  - name: EGRESS_ROUTER_MODE (4)
    value: http-proxy
containers:
- name: egress-router-proxy
  image: registry.redhat.io/openshift3/ose-egress-http-proxy
  env:
  - name: EGRESS_HTTP_PROXY_DESTINATION (5)
    value: |
      !*.example.com
      !192.168.1.0/24
      *
```

(1) プライマリーネットワークインターフェイスで Macvlan ネットワークインターフェイスを作成してから、これを Pod のネットワークプロジェクトに移行し、**egress-router** コンテナーを起動します。**"true"** の周りの引用符をそのまま残します。これらを省略すると、エラーが発生します。

(2) ノードが置かれており、クラスター管理者がこの Pod で使用するために予約している物理ネットワークの IP アドレスです。オプションとして、サブネットの長さ **/24** 接尾辞を組み込み、ローカルサブネットへの適切なルートがセットアップされるようにできます。サブネットの長さを指定しない場合、egress ルーターは **EGRESS_GATEWAY** 変数で指定されたホストにのみアクセスでき、サブネットの他のホストにはアクセスできません。

(3) ノード自体で使用されるデフォルトゲートウェイと同じ値。

(4) これは egress ルーターイメージに対し、これが HTTP プロキシーの一部としてデプロイされているため、iptables のリダイレクトルールを設定できないことを示します。

(5) プロキシーの設定方法を指定する文字列または YAML の複数行文字列です。これは、init コンテナーの他の環境変数ではなく、HTTP プロキシーコンテナーの環境変数として指定されることに注意してください。

EGRESS_HTTP_PROXY_DESTINATION 値に以下のいずれかを指定できます。また、* を使用することができます。これはすべてのリモート宛先への接続を許可することを意味します。 設定の各行には、許可または拒否する接続の 1 つのグループを指定します。

- IP アドレス (例: **192.168.1.1**) は該当する IP アドレスへの接続を許可します。
- CIDR 範囲 (例: **192.168.1.0/24**) は CIDR 範囲への接続を許可します。
- ホスト名 (例: **www.example.com**) は該当ホストへのプロキシーを許可します。
- ***.** が先に付けられるドメイン名 (例: ***.example.com**) は該当ドメインおよびそのサブドメインのすべてへのプロキシーを許可します。

- 上記のいずれかに ! を付けると、接続は許可されるのではなく、拒否されます。
- 最後の行が * の場合、拒否されていないすべてのものが許可されます。または、許可されていないすべてのものが拒否されます。

2. egresss ルーターを参照するサービスを作成し、他の Pod が Pod の IP アドレスを見つけられるようにします。

```
apiVersion: v1
kind: Service
metadata:
  name: egress-1
spec:
  ports:
  - name: http-proxy
    port: 8080 (1)
  type: ClusterIP
  selector:
    name: egress-1
```

(1) **http** ポートが常に **8080** に設定されていることを確認します。

3. **http_proxy** または **https_proxy** 変数を設定して、クライアント Pod (egress プロキシー Pod ではない) を HTTP プロキシーを使用するように設定します。

```
...
env:
- name: http_proxy
  value: http://egress-1:8080/ (1)
- name: https_proxy
  value: http://egress-1:8080/
...
```

(1) 手順 2 で作成されたサービス。

注記

すべてのセットアップに **http_proxy** および **https_proxy** 環境変数が必要になる訳ではありません。上記を実行しても作業用セットアップが作成されない場合は、Pod で実行しているツールまたはソフトウェアについてのドキュメントを参照してください。

リダイレクトする egress ルーターの上記の例 と同様に、ConfigMap を使用して **EGRESS_HTTP_PROXY_DESTINATION** を指定することもできます。

8.6.1.5. Egress ルーター DNS プロキシー Pod のデプロイ

DNS プロキシーモードでは、egress ルーターは、トラフィックを独自の IP アドレスから 1 つ以上の宛先 IP アドレスに送信する TCP ベースのサービスの DNS プロキシーとして実行されます。予約されたソース IP アドレスを使用する必要のあるクライアント Pod は、宛先 IP に直接接続するのでなく、egress ルーターに接続するように変更される必要があります。これにより、外部の宛先でトラフィックが既知のソースから送信されているかのように処理されます。

1. 例として以下を使用して Pod を作成します。

```
apiVersion: v1
kind: Pod
metadata:
  name: egress-dns-proxy
  labels:
    name: egress-dns-proxy
  annotations:
    pod.network.openshift.io/assign-macvlan: "true" (1)
spec:
  initContainers:
  - name: egress-router-setup
    image: registry.redhat.io/openshift3/ose-egress-router
    securityContext:
      privileged: true
    env:
    - name: EGRESS_SOURCE (2)
      value: 192.168.12.99/24
    - name: EGRESS_GATEWAY (3)
      value: 192.168.12.1
    - name: EGRESS_ROUTER_MODE (4)
      value: dns-proxy
  containers:
  - name: egress-dns-proxy
    image: registry.redhat.io/openshift3/ose-egress-dns-proxy
    env:
    - name: EGRESS_DNS_PROXY_DEBUG (5)
      value: "1"
    - name: EGRESS_DNS_PROXY_DESTINATION (6)
      value: |
        # Egress routes for Project "Foo", version 5

        80  203.0.113.25

        100 example.com

        8080 203.0.113.26 80

        8443 foobar.com 443
```

(1) **pod.network.openshift.io/assign-macvlan annotation** を使用することで、プライマリーネットワークインターフェイスで Macvlan ネットワークインターフェイスが作成され、これを Pod のネットワーク namespace に移行してから、**egress-router-setup** コンテナーを起動します。**"true"** の周りの引用符をそのまま残します。これらを省略すると、エラーが発生します。

(2) ノードが置かれており、クラスター管理者がこの Pod で使用するために予約している物理ネットワークの IP アドレスです。オプションとして、サブネットの長さ **/24** 接尾辞を組み込み、ローカルサブネットへの適切なルートがセットアップされるようにできます。サブネットの長さを指定しない場合、egress ルーターは **EGRESS_GATEWAY** 変数で指定されたホストにのみアクセスでき、サブネットの他のホストにはアクセスできません。

(3) ノード自体で使用されるデフォルトゲートウェイと同じ値。

4 これは egress ルーターイメージに対し、これが DNS プロキシーの一部としてデプロイされているため、iptables のリダイレクトルールを設定できないことを示します。

5 オプション。この変数を設定すると、DNS プロキシーログ出力が 標準出力 (stdout) で表示されます。

6 ここでは、複数行の文字列に YAML 構文を使用しています。詳細は以下を参照してください。

注記

EGRESS_DNS_PROXY_DESTINATION の各行は、以下の 2 つの方法のいずれかで設定できます。

- **<port> <remote_address>**: これは、指定の **<port>** への受信接続が指定の **<remote_address>** の同じ TCP ポートにプロキシー送信される必要があることを示しています。**<remote_address>** は IP アドレスまたは DNS 名を指定できます。DNS 名の場合、DNS 解決は起動時に行われます。上記の例では、最初の行はローカルポート 80 から 203.0.113.25 のポート 80 に TCP トラフィックをプロキシー送信します。2 つ目の行は、TCP トラフィックをローカルポート 100 から example.com のポート 100 にプロキシー送信しています。

- **<port> <remote_address> <remote_port>**: 接続が **<remote_address>** の別の **<remote_port>** にプロキシー送信されるのを除き、上記と同じになります。この例では、3 番目の行ではローカルポート 8080 を 203.0.113.26 のリモートポート 80 にプロキシー送信し、4 番目の行ではローカルポート 8443 を footbar.com のリモートポートおよび 443 にプロキシー送信しています。

2. egresss ルーターを参照するサービスを作成し、他の Pod が Pod の IP アドレスを見つけられるようにします。

```
apiVersion: v1
kind: Service
metadata:
  name: egress-dns-svc
spec:
  ports:
  - name: con1
    protocol: TCP
    port: 80
    targetPort: 80
  - name: con2
    protocol: TCP
    port: 100
    targetPort: 100
  - name: con3
    protocol: TCP
    port: 8080
    targetPort: 8080
  - name: con4
    protocol: TCP
    port: 8443
    targetPort: 8443
```

```
  type: ClusterIP
  selector:
    name: egress-dns-proxy
```

Pod がこのサービスに接続できるようになります。これらの接続は、予約された egress IP アドレスを使用して外部サーバーの対応するポートにプロキシー送信されます。

リダイレクトする egress ルーターの上記の例 と同様に、ConfigMap を使用して **EGRESS_DNS_PROXY_DESTINATION** を指定することもできます。

8.6.1.6. Egress ルーター Pod のフェイルオーバーの有効化

レプリケーションコントローラーを使用し、ダウンタイムを防ぐために egress ルーター Pod の 1 つのコピーを常に確保できるようにします。

1. 以下を使用してレプリケーションコントローラーの設定ファイルを作成します。

```
apiVersion: v1
kind: ReplicationController
metadata:
  name: egress-demo-controller
spec:
  replicas: 1 (1)
  selector:
    name: egress-demo
  template:
    metadata:
      name: egress-demo
      labels:
        name: egress-demo
      annotations:
        pod.network.openshift.io/assign-macvlan: "true"
    spec:
      initContainers:
      - name: egress-demo-init
        image: registry.redhat.io/openshift3/ose-egress-router
        env:
        - name: EGRESS_SOURCE (2)
          value: 192.168.12.99/24
        - name: EGRESS_GATEWAY
          value: 192.168.12.1
        - name: EGRESS_DESTINATION
          value: 203.0.113.25
        - name: EGRESS_ROUTER_MODE
          value: init
        securityContext:
          privileged: true
      containers:
      - name: egress-demo-wait
        image: registry.redhat.io/openshift3/ose-pod
      nodeSelector:
        site: springfield-1
```

1. 特定の **EGRESS_SOURCE** 値を使用できる Pod は常に 1 つだけであるため、 **replicas** が **1** に設定されていることを確認します。これは、ルーターの単一コピーのみがラベル **site=springfield-1** が設定されたノードで実行されることを意味します。

2. ノードが置かれており、クラスター管理者がこの Pod で使用するために予約している物理ネットワークの IP アドレスです。オプションとして、サブネットの長さ **/24** 接尾辞を組み込み、ローカルサブネットへの適切なルートがセットアップされるようにできます。サブネットの長さを指定しない場合、egress ルーターは **EGRESS_GATEWAY** 変数で指定されたホストにのみアクセスでき、サブネットの他のホストにはアクセスできません。

2. 定義を使用して Pod を作成します。

```
$ oc create -f <replication_controller>.json
```

3. 検証するには、レプリケーションコントローラー Pod が作成されているかどうかを確認します。

```
$ oc describe rc <replication_controller>
```

8.6.2. 外部リソースへのアクセスを制限するための iptables ルールの使用

クラスター管理者の中には、**EgressNetworkPolicy** のモデルや egress ルーターの対象外の発信トラフィックに対してアクションを実行する必要のある管理者がいる場合があります。この場合には、iptables ルールを直接作成してこれを実行することができます。

たとえば、特定の宛先へのトラフィックをログに記録するルールを作成したり、1 秒ごとに設定される特定数を超える発信接続を許可しないようにしたりできます。

OpenShift Container Platform はカスタム iptables ルールを自動的に追加する方法を提供していませんが、管理者がこのようなルールを手動で追加できる場所を提供します。各ノードは起動時に、**filter** テーブルに **OPENSHIFT-ADMIN-OUTPUT-RULES** という空のチェーンを作成します (チェーンがすでに存在していないと仮定します)。管理者がこのチェーンに追加するすべてのルールは、Pod からクラスター外にある宛先へのすべてのトラフィックに適用されます (それ以外のトラフィックには適用されません)。

この機能を使用する際には、注意すべきいくつかの点があります。

1. 各ノードにルールが作成されていることを確認するのは管理者のタスクになります。OpenShift Container Platform はこれを自動的に確認する方法は提供しません。

2. ルールは egress ルーターによってクラスターを退出するトラフィックには適用されず、ルールは **EgressNetworkPolicy** ルールが適用された後に実行されます (そのため、**EgressNetworkPolicy** で拒否されるトラフィックは表示されません)。

3. ノードには外部 IP アドレスと内部 SDN IP アドレスの両方があるため、Pod からノードまたはノードからマスターへの接続の処理は複雑になります。そのため、一部の Pod とノード間/Pod とマスター間のトラフィックはこのチェーンを通過しますが、他の Pod とノード間/Pod とマスター間のトラフィックはこれをバイパスする場合があります。

8.7. 外部プロジェクトトラフィックの静的 IP の有効化

クラスター管理者は特定の静的 IP アドレスをプロジェクトに割り当て、トラフィックが外部から容易に識別できるようにできます。これは、トラフィックと特定の宛先に送信するために使用されるデフォルトの egress ルーターとは異なります。

識別可能な IP トラフィックは起点を可視化することで、クラスターのセキュリティーを強化します。これが有効にされると、指定されたプロジェクトからのすべての発信外部接続は同じ固定ソース IP を共有します。 つまり、すべての外部リソースがこのトラフィックを認識できるようになります。

egress ルーターの場合とは異なり、これは **EgressNetworkPolicy** ファイアウォールルールに基づいて実行されます。

注記

クラスターのプロジェクトに静的 IP アドレスを割り当てるには、SDN が **ovs-networkpolicy** または **ovs-multitenant** ネットワークプラグインのいずれかを使用する必要があります。

注記

OpenShift SDN をマルチテナントモードで使用する場合、それらに関連付けられたプロジェクトによって別の namespace に参加している namespace と共に egress IP アドレスを使用することはできません。たとえば、**project1** および **project2** に **oc adm pod-network join-projects --to=project1 project2** コマンドを実行して参加している場合、どちらもプロジェクトも egress IP アドレスを使用できません。詳細は、BZ#1645577 を参照してください。

静的ソース IP を有効にするには、以下を実行します。

1. 必要な IP で **NetNamespace** を更新します。

```
$ oc patch netnamespace <project_name> -p '{"egressIPs": ["<IP_address>"]}'
```

たとえば、**MyProject** プロジェクトを IP アドレス 192.168.1.100 に割り当てるには、以下を実行します。

```
$ oc patch netnamespace MyProject -p '{"egressIPs": ["192.168.1.100"]}'
```

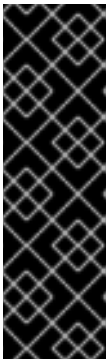
egressIPs フィールドは配列です。**egressIPs** を異なるノードの 2 つ以上の IP アドレスに設定し、高可用性を確保することができます。複数の egress IP アドレスが設定される場合、Pod は egress の一覧にある最初の IP を使用しますが、IP アドレスをホストするノードが失敗する場合、Pod は短時間の遅延の後に一覧にある次の IP の使用に切り替えます。

2. egress IP を必要なノードホストに手動で割り当てます。ノードホストの **HostSubnet** オブジェクトの **egressIPs** フィールドを設定します。そのノードホストに割り当てる必要のある任意の数の IP を含めることができます。

```
$ oc patch hostsubnet <node_name> -p \
  '{"egressIPs": ["<IP_address_1>", "<IP_address_2>"]}'
```

たとえば、**node1** に egress IP 192.168.1.100、192.168.1.101、および 192.168.1.102 が必要な場合は、以下のようになります。

```
$ oc patch hostsubnet node1 -p \
  '{"egressIPs": ["192.168.1.100", "192.168.1.101", "192.168.1.102"]}'
```



重要

egress IP はプライマリーネットワークインターフェイスの追加の IP アドレスとして実装され、ノードのプライマリー IP と同じサブネットに置かれる必要があります。さらに、外部 IP は、**ifcfg-eth0** などの Linux ネットワーク設定ファイルで設定することはできません。

一部のクラウドまたは仮想マシンソリューションを使用する場合に、プライマリーネットワークインターフェイスで追加の IP アドレスを許可するには追加の設定が必要になる場合があります。

プロジェクトに対して上記が有効にされる場合、そのプロジェクトからのすべての egress トラフィックはその egress IP をホストするノードにルーティングされ、(NAT を使用して) その IP アドレスに接続されます。**egressIPs** が **NetNamespace** で設定されているものの、その egress IP をホストするノードがない場合、namespace からの egress トラフィックはドロップされます。

8.8. 自動 EGRESS IP の有効化

外部プロジェクトトラフィックの静的 IP の有効化 の場合のように、クラスター管理者として、**egressIPs** パラメーターを **NetNamespace** リソースに設定して egress IP アドレスを namespace に割り当てることができます。単一 IP アドレスのみをプロジェクトに関連付けることができます。

注記

OpenShift SDN をマルチテナントモードで使用する場合、それらに関連付けられたプロジェクトによって別の namespace に参加している namespace と共に egress IP アドレスを使用することはできません。たとえば、**project1** および **project2** に **oc adm pod-network join-projects --to=project1 project2** コマンドを実行して参加している場合、どちらもプロジェクトも egress IP アドレスを使用できません。詳細は、BZ#1645577 を参照してください。

完全に自動化された egress IP により、各ノードの **HostSubnet** リソースの **egressCIDRs** パラメーターを設定し、ホストできる egress IP アドレスの範囲を示唆することができます。egress IP アドレスを要求した namespace はそれらの egress IP アドレスをホストできるノードに一致し、その後に egress IP アドレスはそれらのノードに割り当てられます。

高可用性は自動的に実行されます。egress IP アドレスをホストしているノードが停止し、**HostSubnet** リソースの **egressCIDR** 値をベースとしてそれらの egress IP アドレスをホストできるノードがある場合、egress IP アドレスは新規ノードに移行します。元の egress IP アドレスノードが再びオンラインに戻ると、ノード間で egress IP アドレスのバランスを図るために egress IP アドレスは自動的に移行します。

重要

手動で割り当てられた egress IP アドレスと自動的に割り当てられた egress IP アドレスの両方を同じノードで使用したり、同じ IP アドレス範囲で使用したりすることはできません。

1. **NetNamespace** を egress IP アドレスで更新します。

```
$ oc patch netnamespace <project_name> -p '{"egressIPs": ["<IP_address>"]}'
```

egressIPs パラメーターに指定できる IP アドレスは 1 つだけです。複数の IP アドレスの使用はサポートされていません。

たとえば、**project1** を IP アドレスの 192.168.1.100 に、 **project2** を IP アドレスの 192.168.1.101 に割り当てるには、以下を実行します。

```
$ oc patch netnamespace project1 -p '{"egressIPs": ["192.168.1.100"]}'
$ oc patch netnamespace project2 -p '{"egressIPs": ["192.168.1.101"]}'
```

2. **egressCIDRs** フィールドを設定して、egress IP アドレスをホストできるノードを示します。

```
$ oc patch hostsubnet <node_name> -p \
  '{"egressCIDRs": ["<IP_address_range_1>", "<IP_address_range_2>"]}'
```

たとえば、**node1** および **node2** を、192.168.1.0 から 192.168.1.255 の範囲で egress IP アドレスをホストするように設定するには、以下を実行します。

```
$ oc patch hostsubnet node1 -p '{"egressCIDRs": ["192.168.1.0/24"]}'
$ oc patch hostsubnet node2 -p '{"egressCIDRs": ["192.168.1.0/24"]}'
```

3. OpenShift Container Platform は、バランスを取りながら特定の egress IP アドレスを利用可能なノードに自動的に割り当てます。この場合、egress IP アドレス 192.168.1.100 を **node1** に、egress IP アドレス 192.168.1.101 を **node2** に割り当て、その逆も行います。

8.9. マルチキャストの有効化

重要

現時点で、マルチキャストは低帯域幅の調整またはサービスの検出での使用に最も適しており、高帯域幅のソリューションとしては適していません。

OpenShift Container Platform の Pod 間のマルチキャストトラフィックはデフォルトで無効にされます。**ovs-multitenant** または **ovs-networkpolicy** プラグインを使用している場合、アノテーションをプロジェクトの対応する **netnamespace** オブジェクトに設定して、プロジェクトごとにマルチキャストを有効にできます。

```
$ oc annotate netnamespace <namespace> \
    netnamespace.network.openshift.io/multicast-enabled=true
```

アノテーションを削除してマルチキャストを無効にします。

```
$ oc annotate netnamespace <namespace> \
    netnamespace.network.openshift.io/multicast-enabled-
```

ovs-multitenant プラグインを使用する場合:

1. 分離したプロジェクトでは、Pod で送信されるマルチキャストパケットはプロジェクト内の他のすべての Pod に送信されます。

2. ネットワークを結合 している場合、すべてのプロジェクトで有効にされるようにマルチキャストを各プロジェクトの **netnamespace** で有効にする必要があります。結合されたネットワークの Pod で送信されるマルチキャストはすべての結合されたネットワークのすべての Pod に送信されます。

3. マルチキャストを **default** プロジェクトで有効にするには、これを **kube-service-catalog** プロ

ジェクトおよび グローバルにされた 他のすべてのプロジェクトで有効にする必要もあります。グローバルプロジェクトの Pod で送信されるマルチキャストパケットはすべてのプロジェクトのすべての Pod にではなく、他のグローバルプロジェクトの Pod に送信されます。同様に、グローバルプロジェクトの Pod はすべてのプロジェクトのすべての Pod からではなく、他のグローバルプロジェクトの Pod から送信されるマルチキャストパケットのみを受信します。

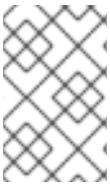
ovs-networkpolicy プラグインを使用する場合:

1. Pod によって送信されるマルチキャストパケットは、**NetworkPolicy** オブジェクトに関係なく、プロジェクトの他のすべての Pod に送信されます。(Pod はユニキャストで通信できない場合でもマルチキャストで通信できます。)

2. 1 つのプロジェクトの Pod によって送信されるマルチキャストパケットは、**NetworkPolicy** オブジェクトがプロジェクト間の通信を許可する場合であっても、それ以外のプロジェクトの Pod に送信されることはありません。

8.10. NETWORKPOLICY の有効化

ovs-subnet および **ovs-multitenant** プラグインにはネットワークの分離についての独自のレガシーモデルがありますが、Kubernetes **NetworkPolicy** はサポートしません。ただし、**NetworkPolicy** サポートは、**ovs-networkpolicy** プラグインを使用すると利用できます。



注記

Egress ポリシータイプ、**ipBlock** パラメーター、および **podSelector** パラメーターと **namespaceSelector** パラメーターを組み合わせる機能は、OpenShift Container Platform では使用できません。

注記

NetworkPolicy 機能はクラスターとの通信に障害を発生させる可能性があるため、これらの機能をデフォルトの OpenShift Container Platform プロジェクトに適用しないでください。

警告

NetworkPolicy ルールは、ホストネットワーク namespace には適用されません。ホストネットワークが有効にされている Pod は **NetworkPolicy** ルールによる影響を受けません。

ovs-networkpolicy プラグインを使用するように設定されている クラスターでは、ネットワークの分離は **NetworkPolicy** オブジェクト によって完全に制御されます。デフォルトで、プロジェクトのすべての Pod は他の Pod およびネットワークのエンドポイントからアクセスできます。プロジェクトで 1 つ以上の Pod を分離するには、そのプロジェクトで **NetworkPolicy** オブジェクトを作成し、許可する着信接続を指定します。プロジェクト管理者は独自のプロジェクト内で **NetworkPolicy** オブジェクトの作成および削除を実行できます。

Pod を参照する **NetworkPolicy** オブジェクトを持たない Pod は完全にアクセスできますが、Pod を参照する 1 つ以上の **NetworkPolicy** オブジェクトを持つ Pod は分離されます。これらの分離された Pod は 1 つ以上の **NetworkPolicy** オブジェクトで許可される接続のみを受け入れます。

複数の異なるシナリオに対応するいくつかの **NetworkPolicy** オブジェクト定義のいくつかを見てみましょう。

- すべてのトラフィックを拒否
  プロジェクトに deny by default (デフォルトで拒否) を実行させるには、すべての Pod に一致するが、トラフィックを一切許可しない **NetworkPolicy** オブジェクトを追加します。

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: deny-by-default
spec:
  podSelector:
  ingress: []
```

- プロジェクト内の **Pod** からの接続のみを許可
  Pod が同じプロジェクト内の他の Pod からの接続を受け入れるが、他のプロジェクトの Pod からの接続を拒否するように設定するには、以下を実行します。

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-same-namespace
spec:
  podSelector:
  ingress:
  - from:
    - podSelector: {}
```

- **Pod** ラベルに基づいて **HTTP** および **HTTPS** トラフィックのみを許可
  特定のラベル (以下の例の **role=frontend**) の付いた Pod への HTTP および HTTPS アクセスのみを有効にするには、以下と同様の **NetworkPolicy** オブジェクトを追加します。

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-http-and-https
spec:
  podSelector:
    matchLabels:
      role: frontend
  ingress:
  - ports:
    - protocol: TCP
      port: 80
    - protocol: TCP
      port: 443
```

NetworkPolicy オブジェクトは加算されるものです。つまり、複数の **NetworkPolicy** オブジェクトを組み合わせて複雑なネットワーク要件を満すことができます。

たとえば、先の例で定義された **NetworkPolicy** オブジェクトの場合、同じプロジェト内に **allow-same-namespace** と **allow-http-and-https** ポリシーの両方を定義することができます。これにより、ラベル **role=frontend** の付いた Pod は各ポリシーで許可されるすべての接続を受け入れます。つま

り、同じ namespace の Pod からのすべてのポート、および すべての namespace の Pod からのポート **80** および **443** での接続を受け入れます。

8.10.1. NetworkPolicy の効率的な使用

NetworkPolicy オブジェクトは、namespace 内でラベルで相互に区別される Pod を分離することを許可します。

NetworkPolicy オブジェクトを単一 namespace 内の多数の個別 Pod に適用することは効率的ではありません。Pod ラベルは IP レベルには存在しないため、**NetworkPolicy** オブジェクトは、**podSelector** で選択されるすべての Pod 間のすべてのリンクについての別個の OVS フロールールを生成します。

たとえば、**spec podSelector** および **ingress podSelector** within a **NetworkPolicy** オブジェクトのそれぞれが 200 Pod に一致する場合、40000 (200*200) OVS フロールールが生成されます。これにより、マシンの速度が低下する可能性があります。

OVS フロールールの量を減らすには、namespace を使用して分離する必要のある Pod のグループを組み込みます。

namespace 全体を選択する **NetworkPolicy** オブジェクトは、**namespaceSelectors** または空の **podSelectors** を使用して、namespace の VXLAN VNID に一致する単一の OVS フロールールのみを生成します。

分離する必要のない Pod は元の namespace に維持し、分離する必要のある Pod は 1 つ以上の異なる namespace に移します。

追加のターゲット設定された namespace 間のポリシーを作成し、分離された Pod から許可する必要のある特定のトラフィックを可能にします。

8.10.2. NetworkPolicy およびルーター

ovs-multitenant プラグインを使用する場合、ルーターからすべての namespace へのトラフィックは自動的に許可されます。これは、ルーターは通常 デフォルトの namespace にあり、すべての namespace がその namespace の Pod からの接続を許可するためです。ただし **ovs-networkpolicy** プラグインを使用すると、これは自動的に実行されません。そのため、デフォルトで namespace を分離するポリシーがある場合は、ルーターがこれにアクセスできるように追加の手順を実行する必要があります。

1 つのオプションとして、すべてのソースからのアクセスを許可する各サービスのポリシーを作成できます。以下は例になります。

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-to-database-service
spec:
  podSelector:
    matchLabels:
      role: database
  ingress:
  - ports:
    - protocol: TCP
      port: 5432
```

これにより、ルーターはサービスにアクセスできますが、同時に他のユーザーの namespace にある Pod もこれにアクセスできます。これらの Pod は通常はパブリックルーターを使用してサービスにアクセスできるため、これによって問題が発生することはないはずです。

または、**ovs-multitenant** プラグインの場合のように、デフォルト namespace からの完全アクセスを許可するポリシーを作成することもできます。

1. ラベルをデフォルト namespace に追加します。

重要

直前の手順でデフォルトプロジェクトに **default** ラベルを付けた場合、この手順を省略します。クラスター管理者ロールは、ラベルを namespace に追加する必要があります。

```
$ oc label namespace default name=default
```

2. その namespace からの接続を許可するポリシーを作成します。

注記

接続を許可するそれぞれの namespace についてこの手順を実行します。プロジェクト管理者ロールを持つユーザーがポリシーを作成できます。

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-from-default-namespace
spec:
  podSelector:
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          name: default
```

8.10.3. 新規プロジェクトのデフォルト NetworkPolicy の設定

クラスター管理者は、新規プロジェクトの作成時に、デフォルトのプロジェクトテンプレートを変更してデフォルトの **NetworkPolicy** オブジェクト (1 つ以上) の自動作成を有効にできます。これを実行するには、以下を行います。

1. カスタムプロジェクトテンプレートを作成し、マスターがこれを使用するように設定します。

2. **default** プロジェクトに **default** ラベルを付けます。

重要

直前の手順でデフォルトプロジェクトに **default** ラベルを付けた場合、この手順を省略します。クラスター管理者ロールは、ラベルを namespace に追加する必要があります。

```
$ oc label namespace default name=default
```

3. 必要な **NetworkPolicy** オブジェクトを含むようにテンプレートを編集します。

```
$ oc edit template project-request -n default
```

注記

NetworkPolicy オブジェクトを既存テンプレートに含めるには、**oc edit** コマンドを使用します。現時点では、**oc patch** を使用してオブジェクトを **Template** リソースに追加することはできません。

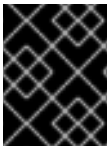
a. それぞれのデフォルトポリシーを **objects** 配列の要素として追加します。

```
objects:
...
- apiVersion: networking.k8s.io/v1
  kind: NetworkPolicy
  metadata:
    name: allow-from-same-namespace
  spec:
    podSelector:
    ingress:
    - from:
      - podSelector: {}
- apiVersion: networking.k8s.io/v1
  kind: NetworkPolicy
  metadata:
    name: allow-from-default-namespace
  spec:
    podSelector:
    ingress:
    - from:
      - namespaceSelector:
          matchLabels:
            name: default
...
```

8.11. HTTP STRICT TRANSPORT SECURITY の有効化

HTTP Strict Transport Security (HSTS) ポリシーは、ホストで HTTPS トラフィックのみを許可するセキュリティーの拡張機能です。デフォルトで、すべての HTTP 要求はドロップされます。これは、web サイトとの対話の安全性を確保したり、ユーザーのためにセキュアなアプリケーションを提供するのに役立ちます。

HSTS が有効にされると、HSTS はサイトから Strict Transport Security ヘッダーを HTTPS 応答に追加します。リダイレクトするルートで **insecureEdgeTerminationPolicy** 値を使用し、HTTP を HTTPS に送信するようにします。ただし、HSTS が有効にされている場合は、要求の送信前にクライアントがすべての要求を HTTP URL から HTTPS に変更するためにリダイレクトの必要がなくなります。これはクライアントでサポートされる必要はなく、**max-age=0** を設定することで無効にできます。

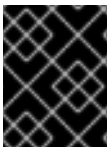


重要

HSTS はセキュアなルート (edge termination または re-encrypt) でのみ機能します。この設定は、HTTP またはパススルールートには適していません。

ルートに対して HSTS を有効にするには、**haproxy.router.openshift.io/hsts_header** 値を edge termination または re-encrypt ルートに追加します。

```
apiVersion: v1
kind: Route
metadata:
  annotations:
    haproxy.router.openshift.io/hsts_header: max-age=31536000;includeSubDomains;preload
```



重要

haproxy.router.openshift.io/hsts_header 値にパラメーターのスペースやその他の値が入っていないことを確認します。**max-age** のみが必要になります。

必須の **max-age** パラメーターは、HSTS ポリシーの有効期間 (秒単位) を示します。クライアントは、ホストから HSTS ヘッダーのある応答を受信する際には常に **max-age** を更新します。**max-age** がタイムアウトになると、クライアントはポリシーを破棄します。

オプションの **includeSubDomains** パラメーターは、クライアントに対し、ホストのすべてのサブドメインがホストど同様に処理されるように指示します。

max-age が 0 より大きい場合、オプションの **preload** パラメーターは外部サービスがこのサイトをそれぞれの HSTS プリロードのリストに含めることを許可します。たとえば、Google などのサイトは **preload** が設定されているサイトの一覧を作成します。ブラウザーはこれらのリストを使用し、サイトと対話する前でも HTTPS 経由でのみ通信するサイトを判別できます。**preload** 設定がない場合、ブラウザーはヘッダーを取得するために HTTPS 経由でサイトと通信している必要があります。

8.12. スループットの問題のトラブルシューティング

OpenShift Container Platform でデプロイされるアプリケーションでは、特定のサービス間で非常に長い待ち時間が発生するなど、ネットワークのスループットの問題が生じることがあります。

Pod のログが問題の原因を指摘しない場合は、以下の方法を使用してパフォーマンスの問題を分析します。

- ping または tcpdump などのパケットアナライザーを使用して Pod とそのノード間のトラフィックを分析します。
  たとえば、問題を生じさせる動作を再現している間に各ノードで tcpdump ツールを実行します。両サイトでキャプチャーしたデータを確認し、送信および受信タイムスタンプを比較して Pod への/からのトラフィックの待ち時間を分析します。待ち時間は、ノードのインターフェイスが他の Pod やストレージデバイス、またはデータプレーンからのトラフィックでオーバーロードする場合に OpenShift Container Platform で発生する可能性があります。

  ```
  $ tcpdump -s 0 -i any -w /tmp/dump.pcap host <podip 1> && host <podip 2>
  ```
  (1)

  (1) **podip** は Pod の IP アドレスです。以下のコマンドを実行して Pod の IP アドレスを取得します。

```
# oc get pod <podname> -o wide
```

tcpdump は、これらの 2 つの Pod 間のすべてのトラフィックが含まれる **/tmp/dump.pcap** のファイルを生成します。理想的には、ファイルサイズを最小限に抑えるために問題を再現するすぐ前と問題を再現したすぐ後ににアナライザーを実行することが良いでしょう。以下のようにノード間でパケットアナライザーを実行することもできます (式から SDN を排除する)。

```
# tcpdump -s 0 -i any -w /tmp/dump.pcap port 4789
```

- ストリーミングのスループットおよび UDP スループットを測定するために iperf などの帯域幅測定ツールを使用します。ボトルネックの特定を試行するには、最初に Pod から、次にノードからツールを実行します。iperf3 ツールは RHEL 7 の一部として組み込まれています。

iperf3 のインストールおよび使用についての詳細は、こちらの Red Hat ソリューション を参照してください。

第9章 サービスアカウントの設定

9.1. 概要

ユーザーが OpenShift Container Platform CLI または web コンソールを使用する場合、API トークンはユーザーを OpenShift Container Platform API に対して認証します。ただし、一般ユーザーの認証情報を利用できない場合、以下のようにコンポーネントが API 呼び出しを行うのが通例になります。以下に例を示します。

- レプリケーションコントローラーが Pod を作成するか、または削除するために API 呼び出しを実行する。
- コンテナー内のアプリケーションが検出目的で API 呼び出しを実行する。
- 外部アプリケーションがモニターまたは統合目的で API 呼び出しを実行する。

サービスアカウントは、一般ユーザーの認証情報を共有せずに API アクセスをより柔軟に制御する方法を提供します。

9.2. ユーザー名およびグループ

すべてのサービスアカウントには、一般ユーザーのようにロールを付与できるユーザー名が関連付けられています。ユーザー名はそのプロジェクトおよび名前から派生します。

```
system:serviceaccount:<project>:<name>
```

たとえば、**view (表示)** ロールを **top-secret** プロジェクトの **robot** サービスアカウントに追加するには、以下を実行します。

```
$ oc policy add-role-to-user view system:serviceaccount:top-secret:robot
```

重要

プロジェクトで特定のサービスアカウントにアクセスを付与する必要がある場合は、**-z** フラグを使用できます。サービスアカウントが属するプロジェクトから **-z** フラグを使用し、**<serviceaccount_name>** を指定します。これによりタイプミスの発生する可能性が減り、アクセスを指定したサービスアカウントのみに付与できるため、この方法を使用することを強くお勧めします。以下に例を示します。

```
$ oc policy add-role-to-user <role_name> -z <serviceaccount_name>
```

プロジェクトから実行しない場合は、以下の例に示すように **-n** オプションを使用してこれが適用されるプロジェクトの namespace を指定します。

すべてのサービスアカウントは 2 つのグループのメンバーでもあります。

system:serviceaccounts
システムのすべてのサービスアカウントが含まれます。

system:serviceaccounts:<project>
指定されたプロジェクトのすべてのサービスアカウントが含まれます。

たとえば、すべてのプロジェクトのすべてのサービスアカウントが **top-secret** プロジェクトのリソースを表示できるようにするには、以下を実行します。

```
$ oc policy add-role-to-group view system:serviceaccounts -n top-secret
```

managers プロジェクトのすべてのサービスアカウントが **top-secret** プロジェクトのリソースを編集できるようにするには、以下を実行します。

```
$ oc policy add-role-to-group edit system:serviceaccounts:managers -n top-secret
```

9.3. サービスアカウントの管理

サービスアカウントは、各プロジェクトに存在する API オブジェクトです。サービスアカウントを管理するには、**sa** または **serviceaccount** オブジェクトタイプと共に **oc** コマンドを使用するか、または web コンソールを使用することができます。

現在のプロジェクトの既存のサービスアカウントの一覧を取得するには、以下を実行します。

```
$ oc get sa
NAME      SECRETS   AGE
builder   2         2d
default   2         2d
deployer  2         2d
```

新規のサービスアカウントを作成するには、以下を実行します。

```
$ oc create sa robot
serviceaccount "robot" created
```

サービスアカウントの作成後すぐに、以下の 2 つのシークレットが自動的に追加されます。

- API トークン
- OpenShift Container レジストリーの認証情報

これらはサービスアカウントを記述すると表示できます。

```
$ oc describe sa robot
Name:  robot
Namespace: project1
Labels:  <none>
Annotations: <none>

Image pull secrets: robot-dockercfg-qzbhb

Mountable secrets:  robot-token-f4khf
                    robot-dockercfg-qzbhb

Tokens:             robot-token-f4khf
                    robot-token-z8h44
```

システムは、サービスアカウントが常に API トークンとレジストリーの認証情報を持っていることを保証します。

生成される API トークンとレジストリーの認証情報は期限切れになることはありませんが、シークレットを削除することで取り消すことができます。シークレットが削除されると、新規のシークレットが自動生成され、これに置き換わります。

9.4. サービスアカウント認証の有効化

サービスアカウントは、プライベート RSA キーで署名されるトークンを使用して API に対して認証されます。認証層では一致するパブリック RSA キーを使用して署名を検証します。

サービスアカウントトークンの生成を有効にするには、マスターで **/etc/origin/master/master-config.yml** ファイルの **serviceAccountConfig** スタンザを更新し、(署名 用に) **privateKeyFile** と **publicKeyFiles** 一覧の一致するパブリックキーファイルを指定します。

```
serviceAccountConfig:
  ...
  masterCA: ca.crt 1
  privateKeyFile: serviceaccount.private.key 2
  publicKeyFiles:
  - serviceaccount.public.key 3
  - ...
```

1. API サーバーの提供する証明書を検証するために使用される CA ファイル。
2. プライベート RSA キーファイル (トークンの署名用)。
3. パブリック RSA キーファイル (トークンの検証用)。プライベートキーファイルが提供されている場合、パブリックキーコンポーネントが使用されます。複数のパブリックキーファイルを使用でき、トークンはパブリックキーのいずれかで検証できる場合に受け入れられます。これにより、署名するキーのローテーションが可能となり、以前の署名者が生成したトークンは依然として受け入れられます。

9.5. 管理サービスアカウント

サービスアカウントは、ビルド、デプロイメントおよびその他の Pod を実行するために各プロジェクトで必要になります。マスターの **/etc/origin/master/master-config.yml** ファイルの **managedNames** 設定は、すべてのプロジェクトに自動作成されるサービスアカウントを制御します。

```
serviceAccountConfig:
  ...
  managedNames: 1
  - builder 2
  - deployer 3
  - default 4
  - ...
```

1. すべてのプロジェクトで自動作成するサービスアカウントの一覧。
2. 各プロジェクトの **builder** サービスアカウントはビルド Pod で必要になり、**system:image-builder** ロールが付与されます。 このロールは、内部コンテナーイメージレジストリーを使用してイメージをプロジェクトのイメージストリームにプッシュすることを可能にします。
3. 各プロジェクトの **deployer** サービスアカウントはデプロイメント Pod で必要になり、レプリケーションコントローラーおよびプロジェクトの Pod の表示および変更を可能にする

system:deployer ロールが付与されます。

4 デフォルトのサービスアカウントは、別のサービスアカウントが指定されない限り、他のすべての Pod で使用されます。

プロジェクトのすべてのサービスアカウントには **system:image-puller** ロールが付与されます。 このロールは、内部コンテナーイメージレジストリーを使用してイメージをイメージストリームからプルすることを可能にします。

9.6. インフラストラクチャーサービスアカウント

一部のインフラストラクチャーコントローラーは、サービスアカウント認証情報を使用して実行されます。以下のサービスアカウントは、サーバーの起動時に OpenShift Container Platform インフラストラクチャープロジェクト (**openshift-infra**) に作成され、クラスター全体で以下のロールが付与されます。

サービスアカウント	説明
replication-controller	system:replication-controller ロールの割り当て
deployment-controller	system:deployment-controller ロールの割り当て
build-controller	system:build-controller ロールの割り当て。さらに、build-controller サービスアカウントは、特権付きの ビルド Pod を作成するために特権付きセキュリティーコンテキストに組み込まれます。

これらのサービスアカウントが作成されるプロジェクトを設定するには、マスターで **/etc/origin/master/master-config.yml** ファイルの **openshiftInfrastructureNamespace** フィールドを設定します。

```
policyConfig:
  ...
  openshiftInfrastructureNamespace: openshift-infra
```

9.7. サービスアカウントおよびシークレット

マスターで **/etc/origin/master/master-config.yml** ファイルの **limitSecretReferences** フィールドを **true** に設定し、Pod のシークレット参照をサービスアカウントでホワイトリストに入れることが必要になるようにします。この値を **false** に設定すると、Pod がプロジェクトのすべてのシークレットを参照できるようになります。

```
serviceAccountConfig:
  ...
  limitSecretReferences: false
```

第10章 ロールベースアクセス制御 (RBAC) の管理

10.1. 概要

CLI を使用して RBAC リソース を表示し、管理者 CLI を使用して ロールとバインディング を管理することができます。

10.2. ロールとバインディングの表示

ロール は、クラスター全体 および プロジェクトのスコープ の両方で各種のアクセスレベルを付与するために使用できます。ユーザーおよびグループ は、1度に複数のロールに関連付けるか、または バインド することができます。**oc describe** コマンドを使用して、ロールおよびそれらのバインディングの詳細を確認できます。

クラスター全体でバインドされた **cluster-admin** のデフォルトクラスターロール を持つユーザーはすべてのリソースに対してすべてのアクションを実行できます。ローカルにバインドされた **admin** のデフォルトクラスターロール を持つユーザーはそのプロジェクト内のロールとバインディングをローカルに管理できます。

注記

Evaluating Authorization セクションで動詞の詳細リストを確認してください。

10.2.1. クラスターロールの表示

クラスターロールおよびそれらの関連付けられたルールセットを表示するには、以下を実行します。

```
$ oc describe clusterrole.rbac
Name:  admin
Labels:  <none>
Annotations: openshift.io/description=A user that has edit rights within the project and can change the
project's membership.
  rbac.authorization.kubernetes.io/autoupdate=true
PolicyRule:
  Resources       Non-Resource URLs Resource Names Verbs
  ---------       ----------------- -------------- -----
  appliedclusterresourcequotas     []   []  [get list watch]
  appliedclusterresourcequotas.quota.openshift.io  []   []  [get list watch]
  bindings       []   []  [get list watch]
  buildconfigs       []   []  [create delete deletecollection get list patch update watch]
  buildconfigs.build.openshift.io    []   []  [create delete deletecollection get list patch update watch]
  buildconfigs/instantiate     []   []  [create]
  buildconfigs.build.openshift.io/instantiate   []   []  [create]
  buildconfigs/instantiatebinary    []   []  [create]
  buildconfigs.build.openshift.io/instantiatebinary  []   []  [create]
  buildconfigs/webhooks      []   []  [create delete deletecollection get list patch update watch]
  buildconfigs.build.openshift.io/webhooks   []   []  [create delete deletecollection get list patch update
watch]
  buildlogs       []   []  [create delete deletecollection get list patch update watch]
  buildlogs.build.openshift.io     []   []  [create delete deletecollection get list patch update watch]
  builds       []   []  [create delete deletecollection get list patch update watch]
  builds.build.openshift.io     []   []  [create delete deletecollection get list patch update watch]
  builds/clone       []   []  [create]
```

```
  builds.build.openshift.io/clone    []   []  [create]
  builds/details      []   []  [update]
  builds.build.openshift.io/details    []   []  [update]
  builds/log       []   []  [get list watch]
  builds.build.openshift.io/log     []   []  [get list watch]
  configmaps       []   []  [create delete deletecollection get list patch update watch]
  cronjobs.batch      []   []  [create delete deletecollection get list patch update watch]
  daemonsets.extensions      []   []  [get list watch]
  deploymentconfigrollbacks     []   []  [create]
  deploymentconfigrollbacks.apps.openshift.io   []   []  [create]
  deploymentconfigs      []   []  [create delete deletecollection get list patch update watch]
  deploymentconfigs.apps.openshift.io    []   []  [create delete deletecollection get list patch update
watch]
  deploymentconfigs/instantiate     []   []  [create]
  deploymentconfigs.apps.openshift.io/instantiate  []   []  [create]
  deploymentconfigs/log      []   []  [get list watch]
  deploymentconfigs.apps.openshift.io/log   []   []  [get list watch]
  deploymentconfigs/rollback     []   []  [create]
  deploymentconfigs.apps.openshift.io/rollback   []   []  [create]
  deploymentconfigs/scale     []   []  [create delete deletecollection get list patch update watch]
  deploymentconfigs.apps.openshift.io/scale   []   []  [create delete deletecollection get list patch
update watch]
  deploymentconfigs/status     []   []  [get list watch]
  deploymentconfigs.apps.openshift.io/status   []   []  [get list watch]
  deployments.apps      []   []  [create delete deletecollection get list patch update watch]
  deployments.extensions     []   []  [create delete deletecollection get list patch update watch]
  deployments.extensions/rollback    []   []  [create delete deletecollection get list patch update watch]
  deployments.apps/scale     []   []  [create delete deletecollection get list patch update watch]
  deployments.extensions/scale     []   []  [create delete deletecollection get list patch update watch]
  deployments.apps/status     []   []  [create delete deletecollection get list patch update watch]
  endpoints       []   []  [create delete deletecollection get list patch update watch]
  events       []   []  [get list watch]
  horizontalpodautoscalers.autoscaling    []   []  [create delete deletecollection get list patch update
watch]
  horizontalpodautoscalers.extensions    []   []  [create delete deletecollection get list patch update
watch]
  imagestreamimages      []   []  [create delete deletecollection get list patch update watch]
  imagestreamimages.image.openshift.io    []   []  [create delete deletecollection get list patch update
watch]
  imagestreamimports      []   []  [create]
  imagestreamimports.image.openshift.io    []   []  [create]
  imagestreammappings      []   []  [create delete deletecollection get list patch update watch]
  imagestreammappings.image.openshift.io   []   []  [create delete deletecollection get list patch update
watch]
  imagestreams       []   []  [create delete deletecollection get list patch update watch]
  imagestreams.image.openshift.io    []   []  [create delete deletecollection get list patch update watch]
  imagestreams/layers      []   []  [get update]
  imagestreams.image.openshift.io/layers   []   []  [get update]
  imagestreams/secrets      []   []  [create delete deletecollection get list patch update watch]
  imagestreams.image.openshift.io/secrets   []   []  [create delete deletecollection get list patch update
watch]
  imagestreams/status      []   []  [get list watch]
  imagestreams.image.openshift.io/status   []   []  [get list watch]
  imagestreamtags      []   []  [create delete deletecollection get list patch update watch]
  imagestreamtags.image.openshift.io    []   []  [create delete deletecollection get list patch update
watch]
```

```
  jenkins.build.openshift.io     []   []  [admin edit view]
  jobs.batch       []   []  [create delete deletecollection get list patch update watch]
  limitranges      []   []  [get list watch]
  localresourceaccessreviews     []   []  [create]
  localresourceaccessreviews.authorization.openshift.io  []   []  [create]
  localsubjectaccessreviews      []   []  [create]
  localsubjectaccessreviews.authorization.k8s.io  []   []  [create]
  localsubjectaccessreviews.authorization.openshift.io  []   []  [create]
  namespaces       []   []  [get list watch]
  namespaces/status      []   []  [get list watch]
  networkpolicies.extensions     []   []  [create delete deletecollection get list patch update watch]
  persistentvolumeclaims     []   []  [create delete deletecollection get list patch update watch]
  pods       []   []  [create delete deletecollection get list patch update watch]
  pods/attach      []   []  [create delete deletecollection get list patch update watch]
  pods/exec      []   []  [create delete deletecollection get list patch update watch]
  pods/log       []   []  [get list watch]
  pods/portforward      []   []  [create delete deletecollection get list patch update watch]
  pods/proxy       []   []  [create delete deletecollection get list patch update watch]
  pods/status      []   []  [get list watch]
  podsecuritypolicyreviews     []   []  [create]
  podsecuritypolicyreviews.security.openshift.io  []   []  [create]
  podsecuritypolicyselfsubjectreviews    []   []  [create]
  podsecuritypolicyselfsubjectreviews.security.openshift.io []   []  [create]
  podsecuritypolicysubjectreviews    []   []  [create]
  podsecuritypolicysubjectreviews.security.openshift.io  []   []  [create]
  processedtemplates      []   []  [create delete deletecollection get list patch update watch]
  processedtemplates.template.openshift.io   []   []  [create delete deletecollection get list patch update
watch]
  projects       []   []  [delete get patch update]
  projects.project.openshift.io     []   []  [delete get patch update]
  replicasets.extensions     []   []  [create delete deletecollection get list patch update watch]
  replicasets.extensions/scale     []   []  [create delete deletecollection get list patch update watch]
  replicationcontrollers     []   []  [create delete deletecollection get list patch update watch]
  replicationcontrollers/scale     []   []  [create delete deletecollection get list patch update watch]
  replicationcontrollers.extensions/scale   []   []  [create delete deletecollection get list patch update
watch]
  replicationcontrollers/status     []   []  [get list watch]
  resourceaccessreviews      []   []  [create]
  resourceaccessreviews.authorization.openshift.io  []   []  [create]
  resourcequotas      []   []  [get list watch]
  resourcequotas/status      []   []  [get list watch]
  resourcequotausages      []   []  [get list watch]
  rolebindingrestrictions     []   []  [get list watch]
  rolebindingrestrictions.authorization.openshift.io  []   []  [get list watch]
  rolebindings       []   []  [create delete deletecollection get list patch update watch]
  rolebindings.authorization.openshift.io   []   []  [create delete deletecollection get list patch update
watch]
  rolebindings.rbac.authorization.k8s.io   []   []  [create delete deletecollection get list patch update
watch]
  roles        []   []  [create delete deletecollection get list patch update watch]
  roles.authorization.openshift.io    []   []  [create delete deletecollection get list patch update watch]
  roles.rbac.authorization.k8s.io    []   []  [create delete deletecollection get list patch update watch]
  routes       []   []  [create delete deletecollection get list patch update watch]
  routes.route.openshift.io     []   []  [create delete deletecollection get list patch update watch]
  routes/custom-host      []   []  [create]
  routes.route.openshift.io/custom-host    []   []  [create]
```

```
  routes/status       []   []  [get list watch update]
  routes.route.openshift.io/status    []   []  [get list watch update]
  scheduledjobs.batch      []   []  [create delete deletecollection get list patch update watch]
  secrets       []   []  [create delete deletecollection get list patch update watch]
  serviceaccounts      []   []  [create delete deletecollection get list patch update watch impersonate]
  services       []   []  [create delete deletecollection get list patch update watch]
  services/proxy      []   []  [create delete deletecollection get list patch update watch]
  statefulsets.apps      []   []  [create delete deletecollection get list patch update watch]
  subjectaccessreviews      []   []  [create]
  subjectaccessreviews.authorization.openshift.io  []   []  [create]
  subjectrulesreviews      []   []  [create]
  subjectrulesreviews.authorization.openshift.io  []   []  [create]
  templateconfigs      []   []  [create delete deletecollection get list patch update watch]
  templateconfigs.template.openshift.io    []   []  [create delete deletecollection get list patch update
watch]
  templateinstances      []   []  [create delete deletecollection get list patch update watch]
  templateinstances.template.openshift.io   []   []  [create delete deletecollection get list patch update
watch]
  templates       []   []  [create delete deletecollection get list patch update watch]
  templates.template.openshift.io    []   []  [create delete deletecollection get list patch update watch]


Name:  basic-user
Labels:  <none>
Annotations: openshift.io/description=A user that can get basic information about projects.
  rbac.authorization.kubernetes.io/autoupdate=true
PolicyRule:
  Resources      Non-Resource URLs Resource Names Verbs
  ---------      ----------------- -------------- -----
  clusterroles      []   []  [get list]
  clusterroles.authorization.openshift.io  []   []  [get list]
  clusterroles.rbac.authorization.k8s.io  []   []  [get list watch]
  projectrequests     []   []  [list]
  projectrequests.project.openshift.io   []   []  [list]
  projects      []   []  [list watch]
  projects.project.openshift.io    []   []  [list watch]
  selfsubjectaccessreviews.authorization.k8s.io  []   []  [create]
  selfsubjectrulesreviews    []   []  [create]
  selfsubjectrulesreviews.authorization.openshift.io []   []  [create]
  storageclasses.storage.k8s.io    []   []  [get list]
  users       []   [~]  [get]
  users.user.openshift.io    []   [~]  [get]


Name:  cluster-admin
Labels:  <none>
Annotations: authorization.openshift.io/system-only=true
  openshift.io/description=A super-user that can perform any action in the cluster. When granted to a
user within a project, they have full control over quota and membership and can perform every
action...
  rbac.authorization.kubernetes.io/autoupdate=true
PolicyRule:
  Resources Non-Resource URLs Resource Names Verbs
  --------- ----------------- -------------- -----
    [*]   []  [*]
  *.*  []   []  [*]
```

```
Name:  cluster-debugger
Labels:  <none>
Annotations: authorization.openshift.io/system-only=true
  rbac.authorization.kubernetes.io/autoupdate=true
PolicyRule:
  Resources Non-Resource URLs Resource Names Verbs
  --------- ----------------- -------------- -----
    [/debug/pprof]  []  [get]
    [/debug/pprof/*] []  [get]
    [/metrics]  []  [get]


Name:  cluster-reader
Labels:  <none>
Annotations: authorization.openshift.io/system-only=true
  rbac.authorization.kubernetes.io/autoupdate=true
PolicyRule:
  Resources       Non-Resource URLs Resource Names Verbs
  ---------       ----------------- -------------- -----
          [*]   []  [get]
  apiservices.apiregistration.k8s.io    []   []  [get list watch]
  apiservices.apiregistration.k8s.io/status   []   []  [get list watch]
  appliedclusterresourcequotas     []   []  [get list watch]

...
```

10.2.2. クラスターのロールバインディングの表示

各種のロールにバインドされたユーザーおよびグループを示す、クラスターのロールバインディングの現在のセットを表示するには、以下を実行します。

```
$ oc describe clusterrolebinding.rbac
Name:  admin
Labels:  <none>
Annotations: rbac.authorization.kubernetes.io/autoupdate=true
Role:
  Kind: ClusterRole
  Name: admin
Subjects:
  Kind   Name    Namespace
  ----   ----    ---------
  ServiceAccount template-instance-controller openshift-infra


Name:  basic-users
Labels:  <none>
Annotations: rbac.authorization.kubernetes.io/autoupdate=true
Role:
  Kind: ClusterRole
  Name: basic-user
Subjects:
  Kind Name   Namespace
  ---- ----   ---------
```

```
  Group system:authenticated


Name:  cluster-admin
Labels:  kubernetes.io/bootstrapping=rbac-defaults
Annotations: rbac.authorization.kubernetes.io/autoupdate=true
Role:
  Kind: ClusterRole
  Name: cluster-admin
Subjects:
  Kind   Name  Namespace
  ----   ----  ---------
  ServiceAccount pvinstaller default
  Group   system:masters


Name:  cluster-admins
Labels:  <none>
Annotations: rbac.authorization.kubernetes.io/autoupdate=true
Role:
  Kind: ClusterRole
  Name: cluster-admin
Subjects:
  Kind Name   Namespace
  ---- ----   ---------
  Group system:cluster-admins
  User system:admin


Name:  cluster-readers
Labels:  <none>
Annotations: rbac.authorization.kubernetes.io/autoupdate=true
Role:
  Kind: ClusterRole
  Name: cluster-reader
Subjects:
  Kind Name   Namespace
  ---- ----   ---------
  Group system:cluster-readers


Name:  cluster-status-binding
Labels:  <none>
Annotations: rbac.authorization.kubernetes.io/autoupdate=true
Role:
  Kind: ClusterRole
  Name: cluster-status
Subjects:
  Kind Name   Namespace
  ---- ----   ---------
  Group system:authenticated
  Group system:unauthenticated


Name:  registry-registry-role
Labels:  <none>
```

```
Annotations: <none>
Role:
  Kind: ClusterRole
  Name: system:registry
Subjects:
  Kind   Name  Namespace
  ----   ----  ---------
  ServiceAccount registry default


Name:  router-router-role
Labels:  <none>
Annotations: <none>
Role:
  Kind: ClusterRole
  Name: system:router
Subjects:
  Kind   Name Namespace
  ----   ---- ---------
  ServiceAccount router default


Name:  self-access-reviewers
Labels:  <none>
Annotations: rbac.authorization.kubernetes.io/autoupdate=true
Role:
  Kind: ClusterRole
  Name: self-access-reviewer
Subjects:
  Kind Name   Namespace
  ---- ----   ---------
  Group system:authenticated
  Group system:unauthenticated


Name:  self-provisioners
Labels:  <none>
Annotations: rbac.authorization.kubernetes.io/autoupdate=true
Role:
  Kind: ClusterRole
  Name: self-provisioner
Subjects:
  Kind Name    Namespace
  ---- ----    ---------
  Group system:authenticated:oauth


Name:  system:basic-user
Labels:  kubernetes.io/bootstrapping=rbac-defaults
Annotations: rbac.authorization.kubernetes.io/autoupdate=true
Role:
  Kind: ClusterRole
  Name: system:basic-user
Subjects:
  Kind Name   Namespace
  ---- ----   ---------
```

```
  Group system:authenticated
  Group system:unauthenticated


Name:  system:build-strategy-docker-binding
Labels:  <none>
Annotations: rbac.authorization.kubernetes.io/autoupdate=true
Role:
  Kind: ClusterRole
  Name: system:build-strategy-docker
Subjects:
  Kind Name   Namespace
  ---- ----   ---------
  Group system:authenticated


Name:  system:build-strategy-jenkinspipeline-binding
Labels:  <none>
Annotations: rbac.authorization.kubernetes.io/autoupdate=true
Role:
  Kind: ClusterRole
  Name: system:build-strategy-jenkinspipeline
Subjects:
  Kind Name   Namespace
  ---- ----   ---------
  Group system:authenticated


Name:  system:build-strategy-source-binding
Labels:  <none>
Annotations: rbac.authorization.kubernetes.io/autoupdate=true
Role:
  Kind: ClusterRole
  Name: system:build-strategy-source
Subjects:
  Kind Name   Namespace
  ---- ----   ---------
  Group system:authenticated


Name:  system:controller:attachdetach-controller
Labels:  kubernetes.io/bootstrapping=rbac-defaults
Annotations: rbac.authorization.kubernetes.io/autoupdate=true
Role:
  Kind: ClusterRole
  Name: system:controller:attachdetach-controller
Subjects:
  Kind   Name   Namespace
  ----   ----   ---------
  ServiceAccount attachdetach-controller kube-system


Name:  system:controller:certificate-controller
Labels:  kubernetes.io/bootstrapping=rbac-defaults
Annotations: rbac.authorization.kubernetes.io/autoupdate=true
Role:
```

```
  Kind: ClusterRole
  Name: system:controller:certificate-controller
Subjects:
  Kind   Name   Namespace
  ----   ----   ---------
  ServiceAccount certificate-controller kube-system


Name:  system:controller:cronjob-controller
Labels:  kubernetes.io/bootstrapping=rbac-defaults
Annotations: rbac.authorization.kubernetes.io/autoupdate=true

...
```

10.2.3. ローカルのロールおよびバインディングの表示

すべての デフォルトクラスターロール は、ユーザーまたはグループにローカルにバインドできます。

カスタムローカルロール を作成できます。

ローカルのロールバインディングも表示することができます。

各種のロールにバインドされたユーザーおよびグループを示す、ローカルのロールバインディングの現在のセットを表示するには、以下を実行します。

```
$ oc describe rolebinding.rbac
```

デフォルトでは、ローカルのロールバインディングを表示する際に現在のプロジェクトが使用されます。または、プロジェクトは **-n** フラグで指定できます。これは、ユーザーに **admin** の デフォルトクラスターロール がすでにある場合、別のプロジェクトのローカルのロールバンディングを表示するのに役立ちます。

```
$ oc describe rolebinding.rbac -n joe-project
Name:  admin
Labels:  <none>
Annotations: <none>
Role:
  Kind: ClusterRole
  Name: admin
Subjects:
  Kind Name Namespace
  ---- ---- ---------
  User joe


Name:  system:deployers
Labels:  <none>
Annotations: <none>
Role:
  Kind: ClusterRole
  Name: system:deployer
Subjects:
  Kind   Name  Namespace
  ----   ----  ---------
  ServiceAccount deployer joe-project
```

```
Name:  system:image-builders
Labels:  <none>
Annotations: <none>
Role:
  Kind: ClusterRole
  Name: system:image-builder
Subjects:
  Kind   Name Namespace
  ----   ---- ---------
  ServiceAccount builder joe-project


Name:  system:image-pullers
Labels:  <none>
Annotations: <none>
Role:
  Kind: ClusterRole
  Name: system:image-puller
Subjects:
  Kind Name     Namespace
  ---- ----     ---------
  Group system:serviceaccounts:joe-project
```

10.3. ロールバインディングの管理

ロール を ユーザーまたはグループ に追加するか、または バインド することにより、そのユーザーまたはグループにそのロールによって付与される関連アクセスが提供されます。**oc adm policy** コマンドを使用して、ユーザーおよびグループに対するロールの追加および削除を実行できます。

以下の操作を使用し、ローカルのロールバインディングでのユーザーまたはグループの関連付けられたロールを管理する際に、プロジェクトは **-n** フラグで指定できます。これが指定されていない場合には、現在のプロジェクトが使用されます。

表10.1 ローカルのロールバインディング操作

コマンド	説明
$ oc adm policy who-can <verb> <resource>	リソースに対してアクションを実行できるユーザーを示します。
$ oc adm policy add-role-to-user <role> <username>	指定されたロールを現在のプロジェクトの指定ユーザーにバインドします。
$ oc adm policy remove-role-from-user <role> <username>	現在のプロジェクトの指定ユーザーから指定されたロールを削除します。
$ oc adm policy remove-user <username>	現在のプロジェクトの指定ユーザーとそれらのロールのすべてを削除します。
$ oc adm policy add-role-to-group <role> <groupname>	指定されたロールを現在のプロジェクトの指定グループにバインドします。

コマンド	説明
$ oc adm policy remove-role-from-group <role> <groupname>	現在のプロジェクトの指定グループから指定されたロールを削除します。
$ oc adm policy remove-group <groupname>	現在のプロジェクトの指定グループとそれらのロールのすべてを削除します。
--rolebinding-name=	ユーザーまたはグループに割り当てられたロールバインディング名を保持するために **oc adm policy** コマンドと共に使用できます。

以下の操作を使用して、クラスターのロールバインディングも管理できます。クラスターのロールバインディングは namespace を使用していないリソースを使用するため、**-n** フラグはこれらの操作に使用されません。

表10.2 クラスターのロールバインディング操作

コマンド	説明
$ oc adm policy add-cluster-role-to-user <role> <username>	指定されたロールをクラスターのすべてのプロジェクトの指定ユーザーにバインドします。
$ oc adm policy remove-cluster-role-from-user <role> <username>	指定されたロールをクラスターのすべてのプロジェクトの指定ユーザーから削除します。
$ oc adm policy add-cluster-role-to-group <role> <groupname>	指定されたロールをクラスターのすべてのプロジェクトの指定グループにバインドします。
$ oc adm policy remove-cluster-role-from-group <role> <groupname>	指定されたロールをクラスターのすべてのプロジェクトの指定グループから削除します。
--rolebinding-name=	ユーザーまたはグループに割り当てられたロールバインディング名を保持するために **oc adm policy** コマンドと共に使用できます。

たとえば、以下を実行して **admin** ロールを **joe-project** の **alice** ユーザーに追加できます。

```
$ oc adm policy add-role-to-user admin alice -n joe-project
```

次に、ローカルのロールバインディングを表示し、出力に追加されていることを確認します。

```
$ oc describe rolebinding.rbac -n joe-project
Name:  admin
Labels:  <none>
Annotations: <none>
Role:
  Kind: ClusterRole
  Name: admin
Subjects:
```

```
  Kind Name Namespace
  ---- ---- ---------
  User joe


Name:  admin-0 (1)
Labels:  <none>
Annotations: <none>

Role:
  Kind:  ClusterRole
  Name:  admin
Subjects:
  Kind  Name   Namespace
  ----  ----   ---------
  User  alice (2)


Name:  system:deployers
Labels:  <none>
Annotations: <none>
Role:
  Kind: ClusterRole
  Name: system:deployer
Subjects:
  Kind   Name  Namespace
  ----   ----  ---------
  ServiceAccount deployer joe-project


Name:  system:image-builders
Labels:  <none>
Annotations: <none>
Role:
  Kind: ClusterRole
  Name: system:image-builder
Subjects:
  Kind   Name Namespace
  ----   ---- ---------
  ServiceAccount builder joe-project


Name:  system:image-pullers
Labels:  <none>
Annotations: <none>
Role:
  Kind: ClusterRole
  Name: system:image-puller
Subjects:
  Kind Name     Namespace
  ---- ----     ---------
  Group system:serviceaccounts:joe-project
```

(1) 新規のロールバインディングがデフォルトの名前で作成され、必要に応じてインクリメントされます。変更する既存のロールバインディングを指定するには、ロールをユーザーに追加する際に **--rolebinding-name** オプションを使用します。

2 ユーザー alice が追加されます。

10.4. ローカルロールの作成

プロジェクトのローカルロールを作成し、これをユーザーにバインドできます。

1. プロジェクトのローカルロールを作成するには、以下のコマンドを実行します。

```
$ oc create role <name> --verb=<verb> --resource=<resource> -n <project>
```

このコマンドで、以下を指定します: * **<name>**、ローカルロールの名前 * **<verb>**、ロールに適用する動詞のコンマ区切りの一覧 * **<resource>**、ロールが適用されるリソース * **<project>**、プロジェクト名

+ たとえば、ユーザーが **blue** プロジェクトで Pod を閲覧できるようにするローカルロールを作成するには、以下のコマンドを実行します。

+

```
$ oc create role podview --verb=get --resource=pod -n blue
```

2. 新規ロールをユーザーにバインドするには、以下のコマンドを実行します。

```
$ oc adm policy add-role-to-user podview user2 --role-namespace=blue -n blue
```

10.5. クラスターロールの作成

クラスターロールを作成するには、以下のコマンドを実行します。

```
$ oc create clusterrole <name> --verb=<verb> --resource=<resource>
```

このコマンドで以下を指定します。

- **<name>**: ローカルのロール名です。
- **<verb>**: ロールに適用する動詞のコンマ区切りの一覧です。
- **<resource>**: ロールが適用されるリソースです。

たとえば、ユーザーが Pod を閲覧できるようにするクラスターロールを作成するには、以下のコマンドを実行します。

```
$ oc create clusterrole podviewonly --verb=get --resource=pod
```

10.6. クラスターおよびローカルのロールバインディング

クラスターのロールバインディングは、クラスターレベルで存在するバインディングです。ロールバインディングは、プロジェクトレベルで存在します。クラスターの **view** (表示) ロールは、ユーザーがプロジェクトを表示できるようローカルのロールバインディングを使用してユーザーにバインドする必要があります。ローカルロールは、クラスターのロールが特定の状況に必要なパーミッションのセットを提供しない場合にのみ作成します。

一部のクラスターのロール名は最初は判別しにくい場合があります。クラスターロール **cluster-admin** は、ローカルのロールバインディングを使用してユーザーにバインドでき、このユーザーがクラスター管理者の特権を持っているように見せます。しかし、実際はそうではありません。**cluster-admin** を特定のプロジェクトにバインドすることにより、そのプロジェクトのスーパーユーザー管理者の場合と同様に、クラスターロール **admin** のパーミッションを付与し、レート制限を編集する機能など、いくつかの追加パーミッションが付与されます。このバインディングは、クラスター管理者にバインドされるクラスターのロールバインディングを一覧表示しない Web コンソール UI を使うと分かりにくくなります。ただし、これは、**cluster-admin** をローカルにバインドするために使用するローカルのロールバインディングを一覧表示します。

10.7. ポリシー定義の更新

クラスターのアップグレード時に、また任意のマスターの再起動時は常に、デフォルトのクラスターロール が欠落しているパーミッションを復元するために自動的に調整されます。

デフォルトクラスターロールをカスタマイズしており、ロールの調整によってそれらが変更されないようにするには、以下を実行します。

1. 各ロールを調整から保護します。

```
$ oc annotate clusterrole.rbac <role_name> --overwrite
rbac.authorization.kubernetes.io/autoupdate=false
```

警告

この設定を含むロールがアップグレード後に新規または必須のパーミッションを組み込むように手動で更新する必要があります。

2. デフォルトのブートストラップポリシーテンプレートを生成します。

```
$ oc adm create-bootstrap-policy-file --filename=policy.json
```

注記

ファイルの内容は OpenShift Container Platform バージョンによって異なりますが、ファイルにはデフォルトポリシーのみが含まれます。

3. **policy.json** ファイルを、クラスターロールのカスタマイズを組み込むように更新します。
4. ポリシーを使用し、調整から保護されていないロールおよびロールバインディングを自動的に調整します。

```
$ oc auth reconcile -f policy.json
```

5. SCC (Security Context Constraints) を調整します。

```
# oc adm policy reconcile-sccs \
    --additive-only=true \
    --confirm
```

第11章 イメージポリシー

11.1. 概要

インポートするイメージや、タグ付けしたり、クラスターで実行したりするイメージを制御することができます。この目的のために使用できる 2 つの機能があります。

インポート用に許可されるレジストリー は、イメージの起点 (origin) を特定の外部レジストリーのセットに制限できるイメージポリシー設定です。このルールセットはイメージストリームにインポートされるか、またはタグ付けされるすべてのイメージに適用されます。したがって、ルールセットと一致しないレジストリーを参照するイメージは拒否されます。

ImagePolicy 受付プラグイン を使用すると、クラスターでの実行を許可するイメージを指定できます。これは現時点ではベータ機能と見なされています。この機能により、以下を制御することができます。

- イメージソース: イメージのプルに使用できるレジストリーについての指定。
- イメージの解決: イメージが再タグ付けによって変更されないよう Pod のイミュータブルなダイジェストでの実行を強制する。
- コンテナーイメージラベルの制限: イメージのラベルを制限するか、または要求する。
- イメージアノテーションの制限: 統合コンテナーイメージレジストリーでイメージのアノテーションを制限するか、または要求する。

11.2. インポート用に許可されるレジストリーの設定

以下の例に示されるように、**imagePolicyConfig:allowedRegistriesForImport** セクション以下にある **master-config.yaml** にインポート用に許可されるレジスターを設定できます。この設定がない場合は、すべてのイメージが許可されます。 これはデフォルトになります。

例11.1 インポート用に許可されるレジストリーの設定例

```
imagePolicyConfig:
  allowedRegistriesForImport:
  -
    domainName: registry.redhat.io (1)
  -
    domainName: *.mydomain.com
    insecure: true (2)
  -
    domainName: local.registry.corp:5000 (3)
```

(1) 指定されたセキュアなレジストリーからのイメージを許可します。

(2) **mydomain.com** の任意のサブドメインでホストされる非セキュアなレジストリーからのイメージを許可します。**mydomain.com** はホワイトリストに追加されません。

(3) ポートが指定された指定レジストリーからのイメージを許可します。

各ルールは以下の属性で設定されています。

- **domainName**: ホスト名であり、オプションでその最後は **:<port>** 接尾辞になり、ここで特殊なワイルドカード文字 (**?**、*) が認識されます。ワイルドカード文字は : 区切り文字の前後の両方に置くことができます。ワイルドカード文字は **:** セパレーターの前後に置くことができます。ワイルドカードは、セパレーターの存在に関係なく、セパレーターの前後の部分にのみ適用されます。

- **insecure**: **:<port>** の部分が **domainName** にない場合、一致するポートを判別するために使用されるブール値です。true の場合、**domainName** はインポート時に非セキュアなフラグが使用されている限り、接尾辞が **:80** のポートが設定されているか、またはポートが未指定のレジストリーに一致します。false の場合、接尾辞が **:443** のポートか、またはポートが未指定のレジストリーが一致します。

ルールが同じドメインのセキュアなポートと非セキュアなポートの両方に一致する場合、ルールは 2 回一覧表示されるはずです (1 回は **insecure=true** が設定され、もう 1 回は **insecure=false** が設定されます)。

修飾されていないイメージ参照は、ルールの評価前に **docker.io** に対して修飾されます。これらをホワイトリストに追加するには、**domainName: docker.io** を使用します。

domainName: * ルールは任意のレジストリーのホスト名に一致しますが、ポートは依然として **443** に制限された状態になります。任意のポートで機能する任意のレジストリーに一致させるには、**domainName: *:*** を使用します。

インポート用に許可されるレジストリーの設定例 で設定されるルールに基づいて、以下が実行されます。

- **oc tag --insecure reg.mydomain.com/app:v1 app:v1** は、**mydomain.com** ルールの処理によってホワイトリストに追加されます。

- **oc import-image --from reg1.mydomain.com:80/foo foo:latest** もホワイトリストに追加されます。

- **oc tag local.registry.corp/bar bar:latest** は、ポートが 3 番目のルールの **5000** に一致しないために拒否されます。

拒否されたイメージのインポートにより、以下のテキストのようなエラーメッセージが生成されます。

```
The ImageStream "bar" is invalid:
* spec.tags[latest].from.name: Forbidden: registry "local.registry.corp" not allowed by whitelist:
"local.registry.corp:5000", "*.mydomain.com:80", "registry.redhat.io:443"
* status.tags[latest].items[0].dockerImageReference: Forbidden: registry "local.registry.corp" not
allowed by whitelist: "local.registry.corp:5000", "*.mydomain.com:80", "registry.redhat.io:443"
```

11.3. IMAGEPOLICY 受付プラグインの設定

クラスターで実行できるイメージを設定するには、ImagePolicy 受付プラグインを **master-config.yaml** ファイルで設定します。1 つまたは複数のルールを必要に応じて設定できます。

- 特定のアノテーションを持つイメージの拒否:
このルールを使用して、特定のアノテーションが設定されたすべてのイメージを拒否します。以下は、**images.openshift.io/deny-execution** アノテーションを使用してすべてのイメージを拒否します。

```
- name: execution-denied
  onResources:
```

```
    - resource: pods
    - resource: builds
    reject: true
    matchImageAnnotations:
    - key: images.openshift.io/deny-execution (1)
      value: "true"
    skipOnResolutionFailure: true
```

(1) 特定のイメージが有害であるとみなされる場合、管理者はそれらのイメージにフラグを付けるためにこのアノテーションを設定できます。

- ユーザーの **Docker Hub** からのイメージの実行を許可:
このルールを使用して、ユーザーが Docker Hub からのイメージを使用できるようにします。

```
- name: allow-images-from-dockerhub
  onResources:
    - resource: pods
    - resource: builds
    matchRegistries:
    - docker.io
```

以下は、複数の ImagePolicy 受付プラグインルールを **master-config.yaml** ファイルに設定する設定例です。

アノテーション付きのサンプルファイル

```
admissionConfig:
  pluginConfig:
    openshift.io/ImagePolicy:
      configuration:
        kind: ImagePolicyConfig
        apiVersion: v1
        resolveImages: AttemptRewrite (1)
        executionRules: (2)
        - name: execution-denied
          # Reject all images that have the annotation images.openshift.io/deny-execution set to true.
          # This annotation may be set by infrastructure that wishes to flag particular images as
dangerous
          onResources: (3)
          - resource: pods
          - resource: builds
          reject: true (4)
          matchImageAnnotations: (5)
          - key: images.openshift.io/deny-execution
            value: "true"
          skipOnResolutionFailure: true (6)
        - name: allow-images-from-internal-registry
          # allows images from the internal registry and tries to resolve them
          onResources:
          - resource: pods
          - resource: builds
          matchIntegratedRegistry: true
        - name: allow-images-from-dockerhub
```

```
    onResources:
    - resource: pods
    - resource: builds
    matchRegistries:
    - docker.io
  resolutionRules: 7
  - targetResource:
      resource: pods
    localNames: true
    policy: AttemptRewrite
  - targetResource: 8
      group: batch
      resource: jobs
    localNames: true 9
    policy: AttemptRewrite
```

1 イミュータブルなイメージダイジェストを使用してイメージを解決し、Pod でイメージのプル仕様を更新します。

2 着信リソースに対して評価するルールの配列です。**reject: true** ルールのみがある場合、デフォルトは **allow all** になります。**reject: false** である accept ルールがルールのいずれかに含まれる場合、ImagePolicy のデフォルト動作は **deny all** に切り替わります。

3 ルールを実施するリソースを示します。何も指定されていない場合、デフォルトは **pods** になります。

4 このルールが一致する場合、Pod は拒否されることを示します。

5 イメージオブジェクトのメタデータで一致するアノテーションの一覧。

6 イメージを解決できない場合に Pod は失敗しません。

7 Kubernetes リソースでのイメージストリームの使用を許可するルールの配列。デフォルト設定は、pods、replicationcontrollers、replicasets、statefulsets、daemonsets、deployments および jobs がイメージフィールドで同じプロジェクトイメージストリームのタグ参照を使用することを許可します。

8 このルールが適用されるグループおよびリソースを特定します。リソースが * の場合、このルールはそのグループのすべてのリソースに適用されます。

9 **LocalNames** は、単一のセグメント名 (例: **ruby:2.5**) が、リソースまたはターゲットイメージストリームで **local name resolution** が有効にされている場合にのみ namespace のローカルイメージストリームタグとして解釈されるようにします。

注記

デフォルトのレジストリー接頭辞 (**docker.io** または **registry.redhat.io** など) を使用してプルされるインフラストラクチャーイメージを通常使用する場合、レジストリー接頭辞がないイメージは **matchRegistries** 値には一致しません。インフラストラクチャーイメージにイメージポリシーに一致するレジストリー接頭辞を持たせるには、**master-config.yaml** ファイルに imageConfig.format 値を設定します。

11.4. 受付コントローラーを使用した ALWAYS PULL イメージの使用

イメージがノードにプルされると、任意のユーザーのノード上の Pod は、イメージに対する承認チェックなしにイメージを使用できます。Pod が認証情報を持たないイメージを使用しないようにするには、**AlwaysPullImages** 受付コントローラーを使用します。

この受付コントローラー はすべての新規 Pod を変更して、イメージプルポリシーを **Always** に強制的に適用し、Pod 仕様が **Never** の イメージプルポリシーを使用する場合でも、プライベートイメージを、それらをプルするための認証情報がある場合のみが使用できるようにします。

AlwaysPullImages 受付コントローラーを有効にするには、以下を実行します。

1. 以下を **master-config.yaml** に追加します。

```
admissionConfig:
  pluginConfig:
    AlwaysPullImages: (1)
      configuration:
        kind: DefaultAdmissionConfig
        apiVersion: v1
        disable: false (2)
```

(1) 受付プラグイン名です。

(2) プラグインを有効にする必要があることを示す **false** を指定します。

2. **master-restart** コマンドを使用してコントロールプレーンの静的 Pod で実行されているマスターサービスを再起動します。

```
$ master-restart api
$ master-restart controllers
```

11.5. IMAGEPOLICY 受付プラグインのテスト

1. **openshift/image-policy-check** を使用して設定をテストします。
たとえば、上記の情報を使用して、以下のようにテストします。

```
$ oc import-image openshift/image-policy-check:latest --confirm
```

2. この YAML を使用して Pod を作成します。Pod が作成されるはずです。

```
apiVersion: v1
kind: Pod
metadata:
  generateName: test-pod
spec:
  containers:
  - image: docker.io/openshift/image-policy-check:latest
    name: first
```

3. 別のレジストリーを参照する別の Pod を作成します。Pod は拒否されます。

```
apiVersion: v1
kind: Pod
metadata:
```

```
  generateName: test-pod
spec:
  containers:
  - image: different-registry/openshift/image-policy-check:latest
    name: first
```

4. インポートされたイメージを使用して、内部レジストリーを参照する Pod を作成します。Pod は作成され、イメージ仕様を確認すると、タグの位置にダイジェストが表示されます。

```
apiVersion: v1
kind: Pod
metadata:
  generateName: test-pod
spec:
  containers:
  - image: <internal registry IP>:5000/<namespace>/image-policy-check:latest
    name: first
```

5. インポートされたイメージを使用して、内部レジストリーを参照する Pod を作成します。Pod は作成され、イメージ仕様を確認すると、タグが変更されていないことを確認できます。

```
apiVersion: v1
kind: Pod
metadata:
  generateName: test-pod
spec:
  containers:
  - image: <internal registry IP>:5000/<namespace>/image-policy-check:v1
    name: first
```

6. **oc get istag/image-policy-check:latest** からダイジェストを取得し、これを **oc annotate images/<digest> images.openshift.io/deny-execution=true** に使用します。以下に例を示します。

```
$ oc annotate
images/sha256:09ce3d8b5b63595ffca6636c7daefb1a615a7c0e3f8ea68e5db044a9340d6ba8
images.openshift.io/deny-execution=true
```

7. この Pod を再作成すると、Pod が拒否されたことがわかります。

```
apiVersion: v1
kind: Pod
metadata:
  generateName: test-pod
spec:
  containers:
  - image: <internal registry IP>:5000/<namespace>/image-policy-check:latest
    name: first
```

第12章 イメージの署名

12.1. 概要

Red Hat Enterprise Linux (RHEL) システムでのコンテナーイメージの署名により、以下を実行できます。

- コンテナーイメージの起点の検証
- イメージが改ざんされていないことの確認
- ホストにプルできる検証済みイメージを判別するポリシーの設定

RHEL システムでのコンテナーイメージの署名についてのアーキテクチャーの詳細は、Container Image Signing Integration Guide を参照してください。

OpenShift Container レジストリーは、REST API 経由で署名を保存する機能を提供します。**oc** CLI を使用して、検証済みのイメージを web コンソールまたは CLI に表示し、イメージの署名を検証することができます。

12.2. ATOMIC CLI を使用したイメージの署名

OpenShift Container Platform はイメージの署名を自動化しません。署名には、通常はワークステーションに安全に保存される開発者のプライベート GPG キーが必要になります。本書では、このワークフローについて説明します。

atomic コマンドラインインターフェイス (CLI)(バージョン 1.12.5 以降) は、OpenShift Container レジストリーにプッシュできるコンテナーイメージに署名するためのコマンドを提供します。**atomic** CLI は、Red Hat ベースのディストリビューション (RHEL、Centos、および Fedora) で利用できます。**atomic** CLI は RHEL Atomic Host システムには事前にインストールされます。**atomic** パッケージの RHEL ホストへのインストールについての詳細は、イメージ署名サポートの有効化 を参照してください。

重要

atomic CLI は、**oc login** で認証された証明書を使用します。**atomic** および **oc** コマンドの両方で同じホストの同じユーザーを使用するようにしてください。たとえば、**atomic** CLI を **sudo** として使用する場合、OpenShift Container Platform に **sudo oc login** を使用してログインします。

署名をイメージに割り当てるには、ユーザーに **image-signer** クラスターロールがなければなりません。クラスター管理者は、以下を使用してこれを追加できます。

```
$ oc adm policy add-cluster-role-to-user system:image-signer <user_name>
```

イメージにはプッシュ時に署名できます。

```
$ atomic push [--sign-by <gpg_key_id>] --type atomic <image>
```

署名は、**atomic** トランスポートタイプの引数が指定される際に OpenShift Container Platform に保存されます。詳細は、Signature Transports を参照してください。

atomic CLI を使用してイメージをセットアップし、実行する方法についての詳細は、RHEL Atomic Host Managing Containers: Signing Container Images ドキュメントか、または **atomic push --help** 出力で引数の詳細を参照してください。

atomic CLI および OpenShift Container レジストリーの使用についてのワークフローの特定の例については、Container Image Signing Integration Guide で説明されています。

12.3. OPENSHIFT CLI を使用したイメージ署名の検証

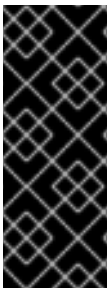
oc adm verify-image-signature コマンドを使用して、OpenShift Container レジストリーにインポートされたイメージの署名を検証できます。このコマンドは、イメージ署名に含まれるイメージ ID が信頼できるかどうかを検証します。 ここでは、パブリック GPG キーを使用して署名自体を検証し、提供される予想 ID と指定イメージの ID (プル仕様) のマッチングが行われます。

デフォルトで、このコマンドは通常 **$GNUPGHOME/pubring.gpg** にあるパブリック GPG キーリングをパス **~/.gnupg** で使用します。デフォルトで、このコマンドは検証結果をイメージオブジェクトに保存し直すことはありません。これを実行するには、以下に示すように **--save** フラグを指定する必要があります。

注記

イメージの署名を検証するには、ユーザーに **image-auditor** クラスターロールがなければなりません。クラスター管理者は、以下を使用してこれを追加できます。

```
$ oc adm policy add-cluster-role-to-user system:image-auditor <user_name>
```



重要

検証済みのイメージで無効な GPG キーまたは無効な予想 ID と共に **--save** フラグを使用すると、保存された検証ステータスおよびすべての署名が削除され、イメージは未検証の状態になります。

誤ってすべての署名を削除してしまうことを避けるために、最初は **--save** フラグなしでコマンドを実行し、ログで潜在的な問題の有無を確認できます。

イメージ署名を検証するには、以下の形式を使用します。

```
$ oc adm verify-image-signature <image> --expected-identity=<pull_spec> [--save] [options]
```

<pull_spec> はイメージストリームを記述することで確認でき、**<image>** はイメージストリームタグを記述して確認することができます。以下のコマンド出力例を参照してください。

イメージ署名の検証例

```
$ oc describe is nodejs -n openshift
Name:            nodejs
Namespace:       openshift
Created:         2 weeks ago
Labels:          <none>
Annotations:     openshift.io/display-name=Node.js
                 openshift.io/image.dockerRepositoryCheck=2017-07-05T18:24:01Z
Docker Pull Spec: 172.30.1.1:5000/openshift/nodejs
...
```

```
$ oc describe istag nodejs:latest -n openshift
Image Name: sha256:2bba968aedb7dd2aafe5fa8c7453f5ac36a0b9639f1bf5b03f95de325238b288
...

$ oc adm verify-image-signature \
    sha256:2bba968aedb7dd2aafe5fa8c7453f5ac36a0b9639f1bf5b03f95de325238b288 \
    --expected-identity 172.30.1.1:5000/openshift/nodejs:latest \
    --public-key /etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release \
    --save
```

注記

oc adm verify-image-signature コマンドが、**x509: certificate signed by unknown authority** エラーを返す場合、レジストリーの認証局 (CA) をシステム上で信頼される CA の一覧に追加する必要がある場合があります。これには、以下の手順を実行します。

1. CA 証明書をクラスターからクライアントマシンに転送します。たとえば、**docker-registry.default.svc** の CA を追加するには、**/etc/docker/certs.d/docker-registry.default.svc\:5000/node-client-ca.crt** にあるファイルを転送します。

2. CA 証明書を **/etc/pki/ca-trust/source/anchors/** ディレクトリーにコピーします。以下に例を示します。

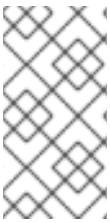
```
# cp </path_to_file>/node-client-ca.crt \
    /etc/pki/ca-trust/source/anchors/
```

3. **update-ca-trust** を実行して、信頼できる CA の一覧を更新します。

```
# update-ca-trust
```

12.4. レジストリー API の使用によるイメージ署名へのアクセス

OpenShift Container レジストリーは、イメージ署名の書き込みおよび読み取りを実行できる **extensions** エンドポイントを提供します。イメージ署名は、コンテナーイメージレジストリー API 経由で OpenShift Container Platform の KVS (key-value store) に保存されます。



注記

このエンドポイントは実験段階にあり、アップストリームのコンテナーイメージレジストリープロジェクトではサポートされていません。コンテナーイメージレジストリー API の一般的な情報については、アップストリーム API のドキュメント を参照してください。

12.4.1. API 経由でのイメージ署名の書き込み

新規署名をイメージに追加するには、HTTP **PUT** メソッドを使用して JSON ペイロードを **extensions** エンドポイントに送信できます。

```
PUT /extensions/v2/<namespace>/<name>/signatures/<digest>
```

```
$ curl -X PUT --data @signature.json http://<user>:
<token>@<registry_endpoint>:5000/extensions/v2/<namespace>/<name>/signatures/sha256:
<digest>
```

署名コンテンツを含む JSON ペイロードの構造は以下のようになります。

```
{
  "version": 2,
  "type":    "atomic",
  "name":
"sha256:4028782c08eae4a8c9a28bf661c0a8d1c2fc8e19dbaae2b018b21011197e1484@cddeb7006d9
14716e2728000746a0b23",
  "content": "<cryptographic_signature>"
}
```

name フィールドには、一意で **<digest>@<name>** 形式でなければならないイメージ署名の名前が含まれます。**<digest>** はイメージ名を表し、**<name>** は署名の名前になります。署名名には 32 文字の長さが必要です。**<cryptographic_signature>** は、コンテナー/イメージ ライブラリーで説明されている仕様に従っている必要があります。

12.4.2. API 経由でのイメージ署名の読み取り

署名済みのイメージが OpenShift Container レジストリーにすでにプッシュされていると仮定すると、以下のコマンドを使って署名を読み取ることができます。

```
GET /extensions/v2/<namespace>/<name>/signatures/<digest>
```

```
$ curl http://<user>:
<token>@<registry_endpoint>:5000/extensions/v2/<namespace>/<name>/signatures/sha256:
<digest>
```

<namespace> は OpenShift Container Platform プロジェクト名またはレジストリーのリポジトリー名を表し、**<name>** はイメージリポジトリーの名前を指します。**digest** はイメージの SHA-256 チェックサムを表します。

指定されたイメージに署名データが含まれる場合、上記のコマンド出力により、以下の JSON 応答が生成されます。

```
{
  "signatures": [
  {
    "version": 2,
    "type":    "atomic",
    "name":
"sha256:4028782c08eae4a8c9a28bf661c0a8d1c2fc8e19dbaae2b018b21011197e1484@cddeb7006d9
14716e2728000746a0b23",
    "content": "<cryptographic_signature>"
  }
  ]
}
```

name フィールドには、一意で **<digest>@<name>** 形式でなければならないイメージ署名の名前が含まれます。**<digest>** はイメージ名を表し、**<name>** は署名の名前になります。署名名には 32 文字の長さが必要です。**<cryptographic_signature>** は、コンテナー/イメージ ライブラリーで説明されている仕

様に従っている必要があります。

12.4.3. 署名ストアからのイメージ署名の自動インポート

OpenShift Container Platform は、署名ストアがすべての OpenShift Container Platform マスターノードに設定されている場合に、レジストリー設定ディレクトリーを使用してイメージ署名を自動インポートします。

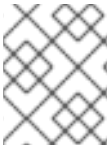
レジストリー設定ディレクトリーには、各種レジストリー (リモートコンテナーイメージを保存するサーバー) およびそれらに保存されるコンテンツの設定が含まれます。この単一ディクトリーを使用すると、設定がコンテナー/イメージのすべてのユーザー間で共有されるように、各コマンドのコマンドラインオプションでその設定を指定する必要がありません。

デフォルトのレジストリー設定ディレクトリーは、**/etc/containers/registries.d/default.yaml** ファイルにあります。

すべての Red Hat イメージに対してイメージ署名の自動インポートを実行する設定例:

```
docker:
  registry.redhat.io:
    sigstore: https://registry.redhat.io/containers/sigstore 1
```

1 署名ストアの URL を定義します。この URL は、既存署名の読み取りに使用されます。



注記

OpenShift Container Platform によって自動的にインポートされる署名は、デフォルトで 未検証 の状態になり、イメージ管理者による検証が必要になります。

レジストリー設定ディレクトリーについての詳細は、Registries Configuration Directory を参照してください。

第13章 スコープ付きトークン

13.1. 概要

ユーザーは、他のエンティティーに対し、自らと同様に機能する権限を制限された方法で付与する必要があるかもしれません。たとえば、プロジェクト管理者は Pod の作成権限を委任する必要があるかもしれません。これを実行する方法の 1 つとして、スコープ付きトークンを作成することができます。

スコープ付きトークンは、指定されるユーザーを識別するが、そのスコープによって特定のアクションに制限するトークンです。現時点で、**cluster-admin** のみがスコープ付きトークンを作成できます。

13.2. 評価

スコープは、トークンの一連のスコープを **PolicyRules** のセットに変換して評価されます。次に、要求がそれらのルールに対してマッチングされます。要求属性は、追加の許可検査のために標準承認者に渡せるよう、スコープルールのいずれかに一致している必要があります。

13.3. ユーザースコープ

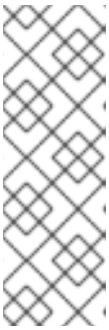
ユーザースコープでは、指定されたユーザーについての情報を取得することにフォーカスが置かれます。それらはインテントベースであるため、ルールは自動的に作成されます。

- **user:full**: ユーザーのすべてのパーミッションによる API の完全な読み取り/書き込みアクセスを許可します。
- **user:info**: ユーザー (名前、グループなど) についての情報の読み取り専用アクセスを許可します。
- **user:check-access**: self-localsubjectaccessreviews および **self-subjectaccessreviews** へのアクセスを許可します。これらは、要求オブジェクトの空のユーザーおよびグループを渡す変数です。
- **user:list-projects**: ユーザーがアクセスできるプロジェクトを一覧表示するための読み取り専用アクセスを許可します。

13.4. ロールスコープ

ロールスコープにより、 namespace でフィルターされる指定ロールと同じレベルのアクセスを持たせることができます。

- **role:<cluster-role name>:<namespace or * for all>**: 指定された namespace のみにあるクラスターロール (cluster-role) で指定されるルールにスコープを制限します。



注記

注意: これは、アクセスのエスカレートを防ぎます。ロールはシークレット、ロールバインディング、およびロールなどのリソースへのアクセスを許可しますが、このスコープはそれらのリソースへのアクセスを制限するのに役立ちます。これにより、予期しないエスカレーションを防ぐことができます。**edit (編集)** などのロールはエスカレートされるロールと見なされないことが多いですが、シークレットのアクセスを持つロールの場合はロールのエスカレーションが生じます。

- **role:<cluster-role name>:<namespace or * for all>:!**: bang (!) を含めることでこのスコープでアクセスのエスカレートを許可されますが、それ以外には上記の例と同様になります。

第14章 イメージのモニターリング

14.1. 概要

CLI を使用して、インスタンスで イメージ および ノード をモニターリングできます。

14.2. イメージ統計の表示

OpenShift Container Platform が管理するすべてのイメージの使用状況についてのいくつかの統計を表示できます。つまり、直接 または ビルド によって内部レジストリーにプッシュされるすべてのイメージの統計を表示できます。

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 images
NAME                IMAGESTREAMTAG            PARENTS                  USAGE
METADATA    STORAGE
sha256:80c985739a78b openshift/python (3.5)                                          yes
303.12MiB
sha256:64461b5111fc7 openshift/ruby (2.2)                                          yes
234.33MiB
sha256:0e19a0290ddc1 test/ruby-ex (latest)     sha256:64461b5111fc71ec   Deployment: ruby-ex-
1/test    yes         150.65MiB
sha256:a968c61adad58 test/django-ex (latest)   sha256:80c985739a78b760   Deployment: django-
ex-1/test  yes         186.07MiB
```

コマンドにより、以下の情報が表示されます。

- イメージ ID
- プロジェクト、名前、および付随する **ImageStreamTag** のタグ
- イメージ ID で一覧表示されるイメージの潜在的な親
- イメージが使用される場所についての情報
- イメージに適切な Docker メタデータ情報が含まれるかどうかを示すフラグ
- イメージのサイズ

14.3. IMAGESTREAMS 統計の表示

すべての **ImageStreams** の使用状況についてのいくつかの統計を表示できます。

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 imagestreams
NAME                STORAGE     IMAGES  LAYERS
openshift/python    1.21GiB     4       36
openshift/ruby      717.76MiB   3       27
test/ruby-ex        150.65MiB   1       10
test/django-ex      186.07MiB   1       10
```

コマンドにより、以下の情報が表示されます。

- プロジェクトおよび **ImageStream** の名前
- 内部 Red Hat Container レジストリー に保存される **ImageStream** 全体のサイズ
- この特定の **ImageStream** が参照しているイメージの数
- **ImageStream** を設定する層の数

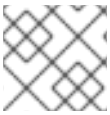
14.4. イメージのプルーニング

上記のコマンドで返される情報は、イメージのプルーニングを実行する際に役立ちます。

第15章 SCC (SECURITY CONTEXT CONSTRAINTS) の管理

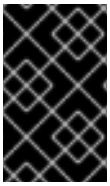
15.1. 概要

SCC (Security Context Constraints) により、管理者は Pod のパーミッションを制御できます。この API タイプについての詳細は、SCC (security context constraints) アーキテクチャーのドキュメントを参照してください。SCC は、CLI を使用し、インスタンスで通常の API オブジェクト として管理できます。



注記

SCC を管理するには、**cluster-admin 権限** が必要です。



重要

デフォルトの SCC を変更しないでください。デフォルトの SCC をカスタマイズすると、アップグレード時に問題が生じる可能性があります。代わりに 新規 SCC を作成 してください。

15.2. SCC (SECURITY CONTEXT CONSTRAINTS) の一覧表示

SCC の現在の一覧を取得するには、以下を実行します。

```
$ oc get scc

NAME              PRIV     CAPS      SELINUX     RUNASUSER        FSGROUP     SUPGROUP
PRIORITY   READONLYROOTFS   VOLUMES
anyuid            false    []        MustRunAs   RunAsAny         RunAsAny    RunAsAny    10
false      [configMap downwardAPI emptyDir persistentVolumeClaim secret]
hostaccess        false    []        MustRunAs   MustRunAsRange   MustRunAs   RunAsAny    <none>
false      [configMap downwardAPI emptyDir hostPath persistentVolumeClaim secret]
hostmount-anyuid  false    []        MustRunAs   RunAsAny         RunAsAny    RunAsAny    <none>
false      [configMap downwardAPI emptyDir hostPath nfs persistentVolumeClaim secret]
hostnetwork       false    []        MustRunAs   MustRunAsRange   MustRunAs   MustRunAs   <none>
false      [configMap downwardAPI emptyDir persistentVolumeClaim secret]
nonroot           false    []        MustRunAs   MustRunAsNonRoot RunAsAny    RunAsAny    <none>
false      [configMap downwardAPI emptyDir persistentVolumeClaim secret]
privileged        true     [*]       RunAsAny    RunAsAny         RunAsAny    RunAsAny    <none>
false      [*]
restricted        false    []        MustRunAs   MustRunAsRange   MustRunAs   RunAsAny    <none>
false      [configMap downwardAPI emptyDir persistentVolumeClaim secret]
```

15.3. SCC (SECURITY CONTEXT CONSTRAINTS) オブジェクトの検査

特定の SCC についての情報 (SCC が適用されるユーザー、サービスアカウントおよびグループを含む) を表示できます。

たとえば、**restricted** SCC を検査するには、以下を実行します。

```
$ oc describe scc restricted
Name:     restricted
Priority:    <none>
```

```
Access:
  Users:    <none> (1)
  Groups:    system:authenticated (2)
Settings:
  Allow Privileged:   false
  Default Add Capabilities:  <none>
  Required Drop Capabilities:  KILL,MKNOD,SYS_CHROOT,SETUID,SETGID
  Allowed Capabilities:   <none>
  Allowed Seccomp Profiles:  <none>
  Allowed Volume Types:
configMap,downwardAPI,emptyDir,persistentVolumeClaim,projected,secret
  Allow Host Network:   false
  Allow Host Ports:   false
  Allow Host PID:   false
  Allow Host IPC:   false
  Read Only Root Filesystem:  false
  Run As User Strategy: MustRunAsRange
    UID:    <none>
    UID Range Min:   <none>
    UID Range Max:   <none>
  SELinux Context Strategy: MustRunAs
    User:    <none>
    Role:    <none>
    Type:    <none>
    Level:    <none>
  FSGroup Strategy: MustRunAs
    Ranges:    <none>
  Supplemental Groups Strategy: RunAsAny
    Ranges:    <none>
```

(1) SCC が適用されるユーザーとサービスアカウントを一覧表示します。

(2) SCC が適用されるグループを一覧表示します。

注記

アップグレード時にカスタマイズされた SCC を保持するには、優先順位、ユーザー、グループ、ラベル、およびアノテーション以外にはデフォルトの SCC の設定を編集しないでください。

15.4. 新規 SCC (SECURITY CONTEXT CONSTRAINTS) の作成

新規 SCC を作成するには、以下を実行します。

1. JSON または YAML ファイルで SCC を定義します。

SCC (Security Context Constraints) オブジェクトの定義

```
kind: SecurityContextConstraints
apiVersion: v1
metadata:
  name: scc-admin
allowPrivilegedContainer: true
runAsUser:
```

```
  type: RunAsAny
seLinuxContext:
  type: RunAsAny
fsGroup:
  type: RunAsAny
supplementalGroups:
  type: RunAsAny
users:
- my-admin-user
groups:
- my-admin-group
```

オプションとして、**requiredDropCapabilities** フィールドに必要な値を設定して、ドロップ機能を SCC に追加することができます。指定された機能はコンテナーからドロップされることになります。たとえば、SCC を **KILL**、**MKNOD**、および **SYS_CHROOT** の必要なドロップ機能を使って作成するには、以下を SCC オブジェクトに追加します。

```
requiredDropCapabilities:
- KILL
- MKNOD
- SYS_CHROOT
```

使用できる値の一覧は、Docker ドキュメント で確認できます。

ヒント

機能は Docker に渡されるため、特殊な **ALL** 値を使用してすべての機能をドロップすることができます。

2. 次に、作成するファイルを渡して **oc create** を実行します。

```
$ oc create -f scc_admin.yaml
securitycontextconstraints "scc-admin" created
```

3. SCC が作成されていることを確認します。

```
$ oc get scc scc-admin
NAME        PRIV    CAPS    SELINUX    RUNASUSER   FSGROUP    SUPGROUP
PRIORITY   READONLYROOTFS   VOLUMES
scc-admin   true    []      RunAsAny   RunAsAny    RunAsAny   RunAsAny   <none>    false
[awsElasticBlockStore azureDisk azureFile cephFS cinder configMap downwardAPI
emptyDir fc flexVolume flocker gcePersistentDisk glusterfs iscsi nfs persistentVolumeClaim
photonPersistentDisk quobyte rbd secret vsphere]
```

15.5. SCC (SECURITY CONTEXT CONSTRAINTS) の削除

SCC を削除するには、以下を実行します。

```
$ oc delete scc <scc_name>
```

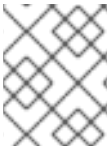
注記

デフォルトの SCC を削除する場合、これは再起動時に再生成されます。

15.6. SCC (SECURITY CONTEXT CONSTRAINTS) の更新

既存 SCC を更新するには、以下を実行します。

```
$ oc edit scc <scc_name>
```



注記

アップグレード時にカスタマイズされた SCC を保持するには、優先順位、ユーザー、グループ以外にデフォルトの SCC の設定を編集しないでください。

15.6.1. SCC (Security Context Constraints) 設定のサンプル

明示的な runAsUser 設定がない場合

```
apiVersion: v1
kind: Pod
metadata:
  name: security-context-demo
spec:
  securityContext: 1
  containers:
  - name: sec-ctx-demo
    image: gcr.io/google-samples/node-hello:1.0
```

1 コンテナーまたは Pod が実行時に使用するユーザー ID を要求しない場合、有効な UID はこの Pod を作成する SCC よって異なります。制限付き SCC はデフォルトですべての認証ユーザーに付与されるため、ほとんどの場合はすべてのユーザーおよびサービスアカウントで利用でき、使用されます。この制限付き SCC は、**securityContext.runAsUser** フィールドの使用できる値を制限し、これをデフォルトに設定するために **MustRunAsRange** ストラテジーを使用します。受付プラグインではこの範囲を指定しないため、現行プロジェクトで **openshift.io/sa.scc.uid-range** アノテーションを検索して範囲フィールドにデータを設定します。最終的にコンテナーの **runAsUser** は予測が困難な範囲の最初の値と等しい値になります。予測が困難であるのはすべてのプロジェクトにはそれぞれ異なる範囲が設定されるためです。詳細は、Understanding Pre-allocated Values and Security Context Constraints (事前に割り当てられた値および SCC (Security Context Constraint) について) を参照してください。

明示的な runAsUser 設定がある場合

```
apiVersion: v1
kind: Pod
metadata:
  name: security-context-demo
spec:
  securityContext:
    runAsUser: 1000 1
```

```
containers:
  - name: sec-ctx-demo
    image: gcr.io/google-samples/node-hello:1.0
```

1 特定のユーザー ID を要求するコンテナーまたは Pod が、OpenShift Container Platform によって受け入れられるのは、サービスアカウントまたはユーザーにそのユーザー ID を許可する SCC へのアクセスが付与されている場合のみです。SCC は、任意の ID や特定の範囲内にある ID、または要求に固有のユーザー ID を許可します。

これは SELinux、fsGroup、および Supplemental Groups で機能します。詳細は、Volume Security (ボリュームセキュリティー) を参照してください。

15.7. デフォルト SCC (SECURITY CONTEXT CONSTRAINTS) の更新

デフォルト SCC は、それらが見つからない場合にはマスターの起動時に作成されます。SCC をデフォルトにリセットするか、またはアップグレード後に既存の SCC を新規のデフォルト定義に更新するには、以下を実行します。

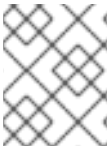
1. リセットする SCC を削除し、マスターを再起動してその再作成を実行します。
2. **oc adm policy reconcile-sccs** コマンドを使用します。

oc adm policy reconcile-sccs コマンドは、すべての SCC ポリシーをデフォルト値に設定しますが、すでに設定した可能性のある追加ユーザー、グループ、ラベル、アノテーションおよび優先順位を保持します。変更される SCC を表示するには、オプションなしでコマンドを実行するか、または **-o <format>** オプションで優先する出力を指定してコマンドを実行します。

確認後は、既存 SCC のバックアップを取ってから **--confirm** オプションを使用してデータを永続化します。

注記

優先順位や許可をリセットする場合は、**--additive-only=false** オプションを使用します。



注記

SCC に優先順位、ユーザー、グループ、ラベル、またはアノテーション以外のカスタマイズ設定がある場合、これらの設定は調整時に失われます。

15.8. 使用方法

以下では、SCC を使用する一般的なシナリオおよび手順について説明します。

15.8.1. 特権付き SCC のアクセス付与

管理者が管理者グループ外のユーザーまたはグループに対して 特権付き **Pod** を追加作成するためのアクセスを付与することが必要になることがあります。これを実行するには、以下を行います。

1. SCC へのアクセスを付与するユーザーまたはグループを決定します。

警告

ユーザーへのアクセス付与は、ユーザーが Pod を直接作成する場合にのみ可能です。ほとんどの場合、システム自体がユーザーの代わりに作成する Pod については、関連するコントローラーの作動に使用される サービスアカウントにアクセスを付与する必要があります。ユーザーの代わりに Pod を作成するリソースの例として、Deployments、StatefulSets、DaemonSets などが含まれます。

2. 以下のコマンドを実行します。

```
$ oc adm policy add-scc-to-user <scc_name> <user_name>
$ oc adm policy add-scc-to-group <scc_name> <group_name>
```

たとえば、**e2e-user** の 特権付き SCC へのアクセスを許可するには、以下を実行します。

```
$ oc adm policy add-scc-to-user privileged e2e-user
```

3. 特権モードを要求するようにコンテナーの **SecurityContext** を変更します。

15.8.2. 特権付き SCC のサービスアカウントアクセスの付与

最初に、サービスアカウント を作成します。たとえば、サービスアカウント **mysvcacct** をプロジェクト **myproject** で作成するには、以下を実行します。

```
$ oc create serviceaccount mysvcacct -n myproject
```

次に、サービスアカウントを **privileged** SCC に追加します。

```
$ oc adm policy add-scc-to-user privileged system:serviceaccount:myproject:mysvcacct
```

その後は、リソースがサービスアカウントの代わりに作成されていることを確認します。これを実行するには、**spec.serviceAccountName** フィールドをサービスアカウント名に設定します。サービスアカウント名を空のままにすると、デフォルト のサービスアカウントが使用されます。

次に、少なくとも 1 つの Pod のコンテナーがセキュリティーコンテキストで特権モードを要求していることを確認します。

15.8.3. Dokerfile の USER によるイメージ実行の有効化

特権付き SCC へのアクセスをすべての人に与えることなく、イメージが事前割り当て UID で強制的に実行されないようにクラスターのセキュリティーを緩和するには、以下を実行します。

1. すべての認証されたユーザーに **anyuid** SCC へのアクセスを付与します。

```
$ oc adm policy add-scc-to-group anyuid system:authenticated
```

警告

これにより、**USER** が **Dockerfile** に指定されていない場合は、イメージをルート ID として実行することができます。

15.8.4. ルートを要求するコンテナーイメージの有効化

一部のコンテナーイメージ (例: **postgres** および **redis**) には root アクセスが必要であり、ボリュームの保有方法についてのいくつかの予測が設定されています。これらのイメージについては、サービスアカウントを **anyuid** SCC に追加します。

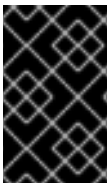
```
$ oc adm policy add-scc-to-user anyuid system:serviceaccount:myproject:mysvcacct
```

15.8.5. レジストリーでの --mount-host の使用

PersistentVolume および **PersistentVolumeClaim** オブジェクトを使用する 永続ストレージ を レジストリーのデプロイメント に使用することが推奨されます。テストを実行中で、**oc adm registry** コマンドを **--mount-host** オプションと共に使用する必要がある場合には、まずレジストリーの新規 サービスアカウント を作成し、これを特権付き SCC に追加する必要があります。詳細の説明については、Administrator Guide を参照してください。

15.8.6. 追加機能の提供

場合によっては、Docker が追加設定なしの機能として提供していない機能がイメージで必要になることがあります。この場合、Pod 仕様で追加機能を要求することができ、これは SCC に対して検証されます。



重要

これによりイメージを昇格された機能を使って実行できますが、これは必要な場合にのみ実行する必要があります。追加機能を有効にするためにデフォルトの **restricted** SCC を編集することはできません。

非 root ユーザーによって使用される場合、**setcap** コマンドを使用して、追加機能を要求するファイルに該当する機能が付与されていることを確認する必要もあります。たとえば、イメージの **Dockerfile** では、以下のようになります。

```
setcap cap_net_raw,cap_net_admin+p /usr/bin/ping
```

さらに機能が Docker のデフォルトとして提供されている場合には、これを要求するために Pod 仕様を変更する必要はありません。たとえば、**NET_RAW** がデフォルトで指定されており、機能がすでに **ping** で設定されている場合、**ping** を実行するのに特別な手順は必要ありません。

追加機能を提供するには、以下を実行します。

1. 新規 SCC を作成します。
2. **allowedCapabilities** フィールドを使用して許可された機能を追加します。

3. Pod の作成時に、**securityContext.capabilities.add** フィールドで機能を要求します。

15.8.7. クラスターのデフォルト動作の変更

すべてのユーザーに対して **anyuid** SCC のアクセスを付与する場合、クラスターは以下のようになります。

- UID を事前に割り当てない
- コンテナーの任意ユーザーとしての実行を許可する
- 特権付きコンテナーを禁止する

```
$ oc adm policy add-scc-to-group anyuid system:authenticated
```

UID を事前に割り当てないようにし、コンテナーが root で実行されないようにクラスターを変更するには、すべてのユーザーに対して **nonroot** SCC のアクセスを付与します。

```
$ oc adm policy add-scc-to-group nonroot system:authenticated
```

警告

クラスター全体に影響を与える変更を行う際には十分に注意してください。直前の例のようにすべての認証ユーザーに SCC を付与したり、制限付き SCC のようにすべてのユーザーに適用される SCC を変更する場合には、Web コンソールや統合コンテナーイメージレジストリーなどの Kubernetes および OpenShift Container Platform コンポーネントにも影響を与えます。これらの SCC に関する変更により、これらのコンポーネントの機能は停止する可能性があります。

代わりに、カスタム SCC を作成し、このターゲットを特定のユーザーまたはグループのみに指定します。これにより、潜在的な問題の影響を特定の影響を受けるユーザーまたはグループに制限し、重要なクラスターコンポーネントに影響が及ばないようにすることができます。

15.8.8. hostPath ボリュームプラグインの使用

privileged、**hostaccess**、または **hostmount-anyuid** などの特権付き SCC へのアクセスをすべての人に付与することなく、Pod で **hostPath** ボリュームプラグインを使用できるようにクラスターのセキュリティーを緩和するには、以下を実行します。

1. **hostpath** という名前の 新規 SCC を作成 します。
2. 新規 SCC の **allowHostDirVolumePlugin** パラメーターを **true** に設定します。

```
$ oc patch scc hostpath -p '{"allowHostDirVolumePlugin": true}'
```

3. この SCC へのアクセスをすべてのユーザーに付与します。

```
$ oc adm policy add-scc-to-group hostpath system:authenticated
```

これで、**hostPath** ボリュームを要求するすべての Pod は **hostpath** SCC で許可されます。

15.8.9. 受付を使用した特定 SCC の初回使用

SCC の **Priority** フィールドを設定し、受付コントローラーで SCC の並び替え順序を制御することができます。並び替えについての詳細は、SCC Prioritization セクションを参照してください。

15.8.10. SCC のユーザー、グループまたはプロジェクトへの追加

SCC をユーザーまたはグループに追加する前に、まず **scc-review** オプションを使用してユーザーまたはグループが Pod を作成できるかどうかをチェックできます。詳細は、Authorization のトピックを参照してください。

SCC はプロジェクトに直接付与されません。代わりに、サービスアカウントを SCC に追加し、Pod にサービスアカウント名を指定するか、または指定されない場合は **default** サービスアカウントを使用して実行します。

SCC をユーザーに追加するには、以下を実行します。

```
$ oc adm policy add-scc-to-user <scc_name> <user_name>
```

SCC をサービスアカウントに追加するには、以下を実行します。

```
$ oc adm policy add-scc-to-user <scc_name> \
    system:serviceaccount:<serviceaccount_namespace>:<serviceaccount_name>
```

現在の場所がサービスアカウントが属するプロジェクトの場合、**-z** フラグを使用し、**<serviceaccount_name>** のみを指定することができます。

```
$ oc adm policy add-scc-to-user <scc_name> -z <serviceaccount_name>
```

重要

上記の **-z** フラグについては、誤字を防ぎ、アクセスが指定されたサービスアカウントのみに付与されるため、使用することを強く推奨します。プロジェクトにいない場合は、**-n** オプションを使用して、それが適用されるプロジェクトの namespace を指定します。

SCC をグループに追加するには、以下を実行します。

```
$ oc adm policy add-scc-to-group <scc_name> <group_name>
```

SCC を namespace のすべてのサービスアカウントに追加するには、以下を実行します。

```
$ oc adm policy add-scc-to-group <scc_name> \
    system:serviceaccounts:<serviceaccount_namespace>
```

第16章 スケジューリング

16.1. 概要

16.1.1. 概要

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。

16.1.2. デフォルトスケジューリング

OpenShift Container Platform 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スケジューラーが同梱されます。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の固有のツールおよびカスタマイズ可能なツールの両方を使用します。

デフォルトスケジューラーが Pod の配置と利用できるカスタマイズ可能なパラメーターを判別する方法についての詳細は、デフォルトスケジューリング を参照してください。

16.1.3. 詳細スケジューリング

新規 Pod の配置場所に対する制御を強化する必要がある場合、OpenShift Container Platform の詳細スケジューリング機能を使用すると、Pod が特定ノード上か、または特定の Pod と共に実行されることを要求する (または実行されることが優先される) よう Pod を設定することができます。また詳細設定により、Pod をノードに配置することや他の Pod と共に実行することを防ぐこともできます。

詳細スケジューリングについての詳細は、詳細スケジューリング を参照してください。

16.1.4. カスタムスケジューリング

OpenShift Container Platform では、Pod 仕様を編集してユーザー独自のスケジューラーまたはサードパーティーのスケジューラーを使用することもできます。

詳細は、カスタムスケジューラー を参照してください。

16.2. デフォルトスケジューリング

16.2.1. 概要

OpenShift Container Platform 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における新規 Pod の配置場所を判別します。スケジューラーは Pod からのデータを読み取り、設定されるポリシーに基づいて適切なノードを見つけようとします。これは完全に独立した機能であり、スタンドアロン/プラグ可能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 のバインディングのみを作成します。

16.2.2. 汎用スケジューラー

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。

1. スケジューラーは 述語を使用して不適切なノードをフィルターに掛けて除外します。
2. スケジューラーは ノードのフィルターされた一覧の優先順位付けを行います。
3. スケジューラーは、Pod の 最も優先順位の高い Pod を選択します。

16.2.3. 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 というフィルター関数の一覧を使用して実行されます。

16.2.3.1. フィルターされたノード一覧の優先順位付け

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、それぞれの優先度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。各優先度関数で指定されるノードのスコアは重み (ほとんどの優先度のデフォルトの重みは 1) で乗算され、すべての優先度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されます。この重み属性は、一部の優先度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。

16.2.3.2. 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。

16.2.4. スケジューラーポリシー

述語 と 優先度 の選択によって、スケジューラーのポリシーが定義されます。

スケジューラー設定ファイルは、スケジューラーが反映する述語と優先度を指定する JSON ファイルです。

スケジューラーポリシーファイルがない場合、デフォルトの設定ファイル **/etc/origin/master/scheduler.json** が適用されます。

重要

スケジューラー設定ファイルで定義される述語および優先度は、デフォルトのスケジューラーポリシーを完全に上書きします。デフォルトの述語および優先度のいずれかが必要な場合、スケジューラー設定ファイルにその関数を明示的に指定する必要があります。

デフォルトのスケジューラー設定ファイル

```
{
    "apiVersion": "v1",
    "kind": "Policy",
    "predicates": [
        {
            "name": "NoVolumeZoneConflict"
        },
        {
```

```
            "name": "MaxEBSVolumeCount"
        },
        {
            "name": "MaxGCEPDVolumeCount"
        },
        {
            "name": "MaxAzureDiskVolumeCount"
        },
        {
            "name": "MatchInterPodAffinity"
        },
        {
            "name": "NoDiskConflict"
        },
        {
            "name": "GeneralPredicates"
        },
        {
            "name": "PodToleratesNodeTaints"
        },
        {
            "argument": {
                "serviceAffinity": {
                    "labels": [
                        "region"
                    ]
                }
            },
            "name": "Region"

        }
    ],
    "priorities": [
        {
            "name": "SelectorSpreadPriority",
            "weight": 1
        },
        {
            "name": "InterPodAffinityPriority",
            "weight": 1
        },
        {
            "name": "LeastRequestedPriority",
            "weight": 1
        },
        {
            "name": "BalancedResourceAllocation",
            "weight": 1
        },
        {
            "name": "NodePreferAvoidPodsPriority",
            "weight": 10000
        },
        {
            "name": "NodeAffinityPriority",
            "weight": 1
```

```
        },
        {
            "name": "TaintTolerationPriority",
            "weight": 1
        },
        {
            "argument": {
                "serviceAntiAffinity": {
                    "label": "zone"
                }
            },
            "name": "Zone",
            "weight": 2
        }
    ]
}
```

16.2.4.1. スケジューラーポリシーの変更

デフォルトで、スケジューラーポリシーはマスター設定ファイルの **kubernetesMasterConfig.schedulerConfigFile** フィールドで上書きされない限り、**/etc/origin/master/scheduler.json** というマスターのファイルの定義されます。

変更されたスケジューラー設定ファイルのサンプル

```
kind: "Policy"
version: "v1"
"predicates": [
        {
            "name": "PodFitsResources"
        },
        {
            "name": "NoDiskConflict"
        },
        {
            "name": "MatchNodeSelector"
        },
        {
            "name": "HostName"
        },
        {
            "argument": {
                "serviceAffinity": {
                    "labels": [
                        "region"
                    ]
                }
            },
            "name": "Region"
        }
    ],
    "priorities": [
        {
            "name": "LeastRequestedPriority",
            "weight": 1
```

```
        },
        {
            "name": "BalancedResourceAllocation",
            "weight": 1
        },
        {
            "name": "ServiceSpreadingPriority",
            "weight": 1
        },
        {
            "argument": {
                "serviceAntiAffinity": {
                    "label": "zone"
                }
            },
            "name": "Zone",
            "weight": 2
        }
    ]
```

スケジューラーポリシーを変更するには、以下を実行します。

1. 必要な デフォルトの述語および優先度 を設定するためにスケジューラー設定ファイルを編集します。カスタム設定を作成したり、サンプルのポリシー設定 のいずれかを使用または変更したりすることができます。
2. 必要な 設定可能な述語 と 設定可能な優先度 を追加します。
3. 変更を有効にするために OpenShift Container Platform を再起動します。

```
# master-restart api
# master-restart controllers
```

16.2.5. 利用可能な述語

述語は、不適切なノードをフィルターに掛けるルールです。

OpenShift Container Platform には、デフォルトでいくつかの述語が提供されています。これらの述語の一部は、特定のパラメーターを指定してカスタマイズできます。複数の述語を組み合わせてノードの追加フィルターを指定できます。

16.2.5.1. 静的な述語

これらの述語はユーザーから設定パラメーターまたは入力を取りません。これらはそれぞれの正確な名前を使用してスケジューラー設定に指定されます。

16.2.5.1.1. デフォルトの述語

デフォルトのスケジューラーポリシーには以下の述語が含まれます。

NoVolumeZoneConflict は、Pod が要求するボリュームがゾーンで利用可能であることを確認します。

```
{"name" : "NoVolumeZoneConflict"}
```

MaxEBSVolumeCount は、AWS インスタンスに割り当てることのできるボリュームの最大数を確認します。

```
{"name" : "MaxEBSVolumeCount"}
```

MaxGCEPDVolumeCount は、Google Compute Engine (GCE) 永続ディスク (PD) の最大数を確認します。

```
{"name" : "MaxGCEPDVolumeCount"}
```

MatchInterPodAffinity は、Pod のアフィニティー/非アフィニティールールが Pod を許可するかどうかを確認します。

```
{"name" : "MatchInterPodAffinity"}
```

NoDiskConflict は、Pod が要求するボリュームが利用可能であるかどうかを確認します。

```
{"name" : "NoDiskConflict"}
```

PodToleratesNodeTaints は Pod がノードのテイントを許容できるかどうかを確認します。

```
{"name" : "PodToleratesNodeTaints"}
```

16.2.5.1.2. 他の静的な述語

OpenShift Container Platform は以下の述語もサポートしています。

CheckVolumeBinding は、バインドされた PVC とバインドされていない PVC の両方について、Pod が要求するボリュームに基づいて Pod が適切かどうかを評価します。* バインドされる PVC の場合、述語は対応する PV のノードアフィニティーが指定ノードで満たされていることを確認します。* バインドされない PVC の場合、述語は PVC 要件を満たし、PV ノードのアフィニティーが指定ノードで満たされる利用可能な PV を検索します。

述語は、すべてのバインドされる PVC にノードと互換性のある PV がある場合や、すべてのバインドされていない PVC が利用可能なノードとの互換性のある PV に一致する場合に true を返します。

```
{"name" : "CheckVolumeBinding"}
```

CheckVolumeBinding 述語は、デフォルト以外のスケジューラーで有効にする必要があります。

CheckNodeCondition は、**out of disk** (ディスク不足)、**network unavailable** (ネットワークが使用不可)、または **not ready** (準備できていない) 状態を報告するノードで Pod をスケジュールできるかどうかを確認します。

```
{"name" : "CheckNodeCondition"}
```

PodToleratesNodeNoExecuteTaints は、Pod の容認がノードの **NoExecute** テイントを容認できるかどうかを確認します。

```
{"name" : "PodToleratesNodeNoExecuteTaints"}
```

CheckNodeLabelPresence は、すべての指定されたラベルがノードに存在するかどうかを確認します (その値が何であるかは問わない)。

```
{"name" : "CheckNodeLabelPresence"}
```

checkServiceAffinity は、ServiceAffinity ラベルがノードでスケジュールされる Pod について同種のものであることを確認します。

```
{"name" : "checkServiceAffinity"}
```

MaxAzureDiskVolumeCount は Azure ディスクボリュームの最大数を確認します。

```
{"name" : "MaxAzureDiskVolumeCount"}
```

16.2.5.2. 汎用的な述語

以下の汎用的な述語は、非クリティカル述語とクリティカル述語が渡されるかどうかを確認します。非クリティカル述語は、非 Critical Pod のみが渡す必要のある述語であり、クリティカル述語はすべての Pod が渡す必要のある述語です。

デフォルトのスケジューラーポリシーには、この汎用的な述語が含まれます。

汎用的な非クリティカル述語
PodFitsResources は、リソースの可用性 (CPU、メモリー、GPU など) に基づいて適切な候補を判別します。ノードはそれらのリソース容量を宣言し、Pod は要求するリソースを指定できます。使用されるリソースではなく、要求されるリソースに基づいて適切な候補が判別されます。

```
{"name" : "PodFitsResources"}
```

必須の汎用的な述語
PodFitsHostPorts は、ノードに要求される Pod ポートの空きポートがある (ポートの競合がない) かどうかを判別します。

```
{"name" : "PodFitsHostPorts"}
```

HostName は、ホストパラメーターの有無と文字列のホスト名との一致に基づいて適切なノードを判別します。

```
{"name" : "HostName"}
```

MatchNodeSelector は、Pod で定義される ノードセレクター (nodeSelector) のクエリーに基づいて適したノードを判別します。

```
{"name" : "MatchNodeSelector"}
```

16.2.5.3. 設定可能な述語

これらの述語はスケジューラー設定 **/etc/origin/master/scheduler.json** (デフォルト) に設定し、述語の機能に影響を与えるラベルを追加することができます。

これらは設定可能であるため、ユーザー定義の名前が異なる限り、同じタイプ (ただし設定パラメーターは異なる) の複数の述語を組み合わせることができます。

これらの優先度の使用方法についての情報は、スケジューラーポリシーの変更 を参照してください。

ServiceAffinity は、Pod で実行されるサービスに基づいて Pod をノードに配置します。同じノードまたは併置されているノードに同じサービスの複数の Pod を配置すると、効率が向上する可能性があります。

この述語は ノードセレクター の特定ラベルを持つ Pod を同じラベルを持つノードに配置しようとします。

Pod がノードセレクターでラベルを指定していない場合、最初の Pod は可用性に基づいて任意のノードに配置され、該当サービスの後続のすべての Pod はそのノードと同じラベルの値を持つノードにスケジュールされます。

```
"predicates":[
    {
      "name":"<name>", 1
      "argument":{
        "serviceAffinity":{
          "labels":[
            "<label>" 2
          ]
        }
      }
    }
  ],
```

1 述語の名前を指定します。

2 一致するラベルを指定します。

以下に例を示します。

```
    "name":"ZoneAffinity",
    "argument":{
        "serviceAffinity":{
            "labels":[
                "rack"
            ]
        }
    }
```

たとえば、ノードセレクター **rack** を持つサービスの最初の Pod がラベル **region=rack** を持つノードにスケジュールされている場合、同じサービスに属するその他すべての後続の Pod は同じ **region=rack** ラベルを持つノードにスケジュールされます。詳細は、Pod 配置の制御 を参照してください。

複数レベルのラベルもサポートされています。ユーザーは同じリージョン内および (リージョン下の) 同じゾーン内のノードでスケジュールされるようサービスのすべての Pod を指定することもできます。

labelsPresence パラメーターは特定のノードに特定のラベルがあるかどうかをチェックします。ラベルは、**LabelPreference** の優先順位が使用するノード グループ を作成します。ラベルでのマッチングは、ノードにラベルで定義されている物理的な場所またはステータスがある場合などに役立ちます。

```
"predicates":[
    {
      "name":"<name>", 1
      "argument":{
```

```
        "labelsPresence":{
          "labels":[
            "<label>" 2
          ],
          "presence": true 3
        }
      }
    }
  ],
```

1 述語の名前を指定します。

2 一致するラベルを指定します。

3 ラベルが必要であるかを、**true** または **false** のいずれかで指定します。

- **presence:false** の場合、要求されるラベルのいずれかがノードラベルにある場合、Pod をスケジュールすることはできません。ラベルが存在しない場合は Pod をスケジュールできます。
- **presence:true** の場合、要求されるラベルのすべてがノードラベルにある場合、Pod をスケジュールできます。ラベルのすべてが存在しない場合、Pod はスケジュールされません。

以下に例を示します。

```
  "name":"RackPreferred",
  "argument":{
    "labelsPresence":{
      "labels":[
        "rack",
        "region"
      ],
      "presence": true
    }
  }
```

16.2.6. 利用可能な優先度

優先度は、設定に応じて残りのノードにランクを付けるルールです。

優先度のカスタムセットは、スケジューラーを設定するために指定できます。OpenShift Container Platform ではデフォルトでいくつかの優先度があります。他の優先度は、特定のパラメーターを指定してカスタマイズできます。優先順位に影響を与えるために、複数の優先度を組み合わせ、異なる重みをそれぞれのノードに指定することができます。

16.2.6.1. 静的優先度

静的優先度は、重みを除き、ユーザーからいずれの設定パラメーターも取りません。重みは指定する必要があり、0 または負の値にすることはできません。

これらはスケジューラー設定 **/etc/origin/master/scheduler.json** (デフォルト) に指定されます。

16.2.6.1.1. デフォルトの優先度

デフォルトのスケジューラーポリシーには、以下の優先度が含まれています。それぞれの優先度関数は、重み **10000** を持つ **NodePreferAvoidPodsPriority** 以外は重み **1** を持ちます。

SelectorSpreadPriority は、Pod に一致するサービス、レプリケーションコントローラー (RC)、レプリケーションセット (RS)、およびステートフルなセットを検索し、次にそれらのセレクターに一致する既存の Pod を検索します。スケジューラーは、一致する既存の Pod が少ないノードを優先します。次に、Pod のスケジュール時に、それらのセレクターに一致する Pod 数の最も少ないノードで Pod をスケジュールします。

```
{"name" : "SelectorSpreadPriority", "weight" : 1}
```

InterPodAffinityPriority は、ノードの対応する PodAffinityTerm が満たされている場合に **weightedPodAffinityTerm** 要素を使った繰り返し処理や 重み の合計への追加によって合計を計算します。合計値の最も高いノードが最も優先されます。

```
{"name" : "InterPodAffinityPriority", "weight" : 1}
```

LeastRequestedPriority は要求されたリソースの少ないノードを優先します。これは、ノードでスケジュールされる Pod によって要求されるメモリーおよび CPU のパーセンテージを計算し、利用可能な/残りの容量の値の最も高いノードを優先します。

```
{"name" : "LeastRequestedPriority", "weight" : 1}
```

BalancedResourceAllocation は、均衡が図られたリソース使用率に基づいてノードを優先します。これは、容量の一部として消費済み CPU とメモリー間の差異を計算し、2 つのメトリクスがどの程度相互に近似しているかに基づいてノードの優先度を決定します。これは常に **LeastRequestedPriority** と併用する必要があります。

```
{"name" : "BalancedResourceAllocation", "weight" : 1}
```

NodePreferAvoidPodsPriority は、レプリケーションコントローラー以外のコントローラーによって所有される Pod を無視します。

```
{"name" : "NodePreferAvoidPodsPriority", "weight" : 10000}
```

NodeAffinityPriority は、ノードアフィニティーのスケジューリング設定に応じてノードの優先順位を決定します。

```
{"name" : "NodeAffinityPriority", "weight" : 1}
```

TaintTolerationPriority は、Pod についての 容認不可能な テイント数の少ないノードを優先します。容認不可能なテイントとはキー **PreferNoSchedule** のあるテイントのことです。

```
{"name" : "TaintTolerationPriority", "weight" : 1}
```

16.2.6.1.2. 他の静的優先度

OpenShift Container Platform は以下の優先度もサポートしています。

EqualPriority は、優先度の設定が指定されていない場合に、すべてのノードに等しい重み **1** を指定します。この優先順位はテスト環境にのみ使用することを推奨します。

```
{"name" : "EqualPriority", "weight" : 1}
```

-

MostRequestedPriority は、要求されたリソースの最も多いノードを優先します。これは、ノードでスケジュールされる Pod で要求されるメモリーおよび CPU のパーセンテージを計算し、容量に対して要求される部分の平均の最大値に基づいて優先度を決定します。

```
{"name" : "MostRequestedPriority", "weight" : 1}
```

ImageLocalityPriority は、Pod コンテナーのイメージをすでに要求しているノードを優先します。

```
{"name" : "ImageLocalityPriority", "weight" : 1}
```

ServiceSpreadingPriority は、同じマシンに置かれる同じサービスに属する Pod 数を最小限にすることにより Pod を分散します。

```
{"name" : "ServiceSpreadingPriority", "weight" : 1}
```

16.2.6.2. 設定可能な優先度

これらの優先度は、デフォルトでスケジューラー設定 **/etc/origin/master/scheduler.json** で設定し、これらの優先度に影響を与えるラベルを追加できます。

優先度関数のタイプは、それらが取る引数によって識別されます。これらは設定可能なため、ユーザー定義の名前が異なる場合に、同じタイプの (ただし設定パラメーターは異なる) 設定可能な複数の優先度を組み合わせることができます。

これらの優先度の使用方法についての情報は、スケジューラーポリシーの変更 を参照してください。

ServiceAntiAffinity はラベルを取り、ラベルの値に基づいてノードのグループ全体に同じサービスに属する Pod を適正に分散します。これは、指定されたラベルの同じ値を持つすべてのノードに同じスコアを付与します。また Pod が最も集中していないグループ内のノードにより高いスコアを付与します。

```
"priorities":[
  {
    "name":"<name>", (1)
    "weight" : 1 (2)
    "argument":{
      "serviceAntiAffinity":{
        "label":[
          "<label>" (3)
        ]
      }
    }
  }
]
```

(1) 優先度の名前を指定します。

(2) 重みを指定します。ゼロ以外の正の値を指定します。

(3) 一致するラベルを指定します。

以下に例を示します。

```
        "name":"RackSpread", (1)
        "weight" : 1 (2)
        "argument":{
            "serviceAntiAffinity":{
                "label": "rack" (3)
            }
        }
```

1. 優先度の名前を指定します。
2. 重みを指定します。ゼロ以外の正の値を指定します。
3. 一致するラベルを指定します。

注記

カスタムラベルに基づいて **ServiceAntiAffinity** を使用しても Pod を予想通りに展開できない場合があります。Red Hat ソリューション を参照してください。

*__labelPreference__ パラメーターは指定されたラベルに基づいて優先順位を指定します。ラベルがノードにある場合、そのノードに優先度が指定されます。ラベルが指定されていない場合、優先度はラベルを持たないノードに指定されます。

```
"priorities":[
    {
        "name":"<name>", (1)
        "weight" : 1, (2)
        "argument":{
            "labelPreference":{
                "label": "<label>", (3)
                "presence": true (4)
            }
        }
    }
]
```

1. 優先度の名前を指定します。
2. 重みを指定します。ゼロ以外の正の値を指定します。
3. 一致するラベルを指定します。
4. ラベルが必要であるかを、**true** または **false** のいずれかで指定します。

16.2.7. 使用例

OpenShift Container Platform 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。

16.2.7.1. インフラストラクチャーのトポロジーレベル

管理者は、ノードのラベル (例: **region=r1**、**zone=z1**、**rack=s1**) を指定してインフラストラクチャーの複数のトポロジーレベルを定義することができます。

これらのラベル名には特別な意味はなく、管理者はそれらのインフラストラクチャーラベルに任意の名前 (例: 都市/建物/部屋) を付けることができます。また、管理者はインフラストラクチャートポロジーに任意の数のレベルを定義できます。通常は、(**regions → zones → racks**) などの 3 つのレベルが適切なサイズです。管理者はこれらのレベルのそれぞれにアフィニティーと非アフィニティールールを任意の組み合わせで指定することができます。

16.2.7.2. 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。同じアフィニティーグループ内で 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に対する制御を強化する必要がある場合は、ノードアフィニティーの使用 および Pod のアフィニティーおよび非アフィニティーの使用 を参照してください。これらの詳細スケジューリング機能により、管理者は Pod をスケジュールできるノードを指定し、他の Pod に関連してスケジューリングを強制的に実行したり、拒否したりできます。

16.2.7.3. 非アフィニティー

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非アフィニティーを設定できるよう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ー (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に対する制御を強化する必要がある場合は、ノードアフィニティーの使用 および Pod のアフィニティーおよび非アフィニティーの使用 を参照してください。これらの詳細スケジューリング機能により、管理者は Pod をスケジュールできるノードを指定し、他の Pod に関連してスケジューリングを強制的に実行したり、拒否したりできます。

16.2.8. ポリシー設定のサンプル

以下の設定は、スケジューラーポリシーファイルを使って指定される場合のデフォルトのスケジューラー設定を示しています。

```
kind: "Policy"
version: "v1"
predicates:
...
  - name: "RegionZoneAffinity" 1
    argument:
      serviceAffinity: 2
        labels: 3
          - "region"
          - "zone"
priorities:
...
  - name: "RackSpread" 4
    weight: 1
```

```
    argument:
      serviceAntiAffinity: 5
        label: "rack" 6
```

1 述語の名前です。

2 述語のタイプ です。

3 述語のラベルです。

4 優先度の名前です。

5 優先度のタイプ です。

6 優先度のラベルです。

以下の設定例のいずれの場合も、述語と優先度関数の一覧は、指定された使用例に関連するもののみを含むように切り捨てられます。実際には、完全な/分かりやすいスケジューラーポリシーには、上記のデフォルトの述語および優先度のほとんど (すべてではなくても) が含まれるはずです。

以下の例は、region (affinity) → zone (affinity) → rack (anti-affinity) の 3 つのトポロジーレベルを定義します。

```
kind: "Policy"
version: "v1"
predicates:
...
  - name: "RegionZoneAffinity"
    argument:
      serviceAffinity:
        labels:
          - "region"
          - "zone"
priorities:
...
  - name: "RackSpread"
    weight: 1
    argument:
      serviceAntiAffinity:
        label: "rack"
```

以下の例は、city (affinity) → building (anti-affinity) → room (anti-affinity) の 3 つのトポロジーレベルを定義します。

```
kind: "Policy"
version: "v1"
predicates:
...
  - name: "CityAffinity"
    argument:
      serviceAffinity:
        labels:
          - "city"
priorities:
...
```

```
  - name: "BuildingSpread"
    weight: 1
    argument:
      serviceAntiAffinity:
        label: "building"
  - name: "RoomSpread"
    weight: 1
    argument:
      serviceAntiAffinity:
        label: "room"
```

以下の例では、region ラベルが定義されたノードのみを使用し、zone ラベルが定義されたノードを優先するポリシーを定義します。

```
kind: "Policy"
version: "v1"
predicates:
...
  - name: "RequireRegion"
    argument:
      labelsPresence:
        labels:
          - "region"
        presence: true
priorities:
...
  - name: "ZonePreferred"
    weight: 1
    argument:
      labelPreference:
        label: "zone"
        presence: true
```

以下の例では、静的および設定可能な述語および優先度を組み合わせています。

```
kind: "Policy"
version: "v1"
predicates:
...
  - name: "RegionAffinity"
    argument:
      serviceAffinity:
        labels:
          - "region"
  - name: "RequireRegion"
    argument:
      labelsPresence:
        labels:
          - "region"
        presence: true
  - name: "BuildingNodesAvoid"
    argument:
      labelsPresence:
        labels:
          - "building"
```

```
      presence: false
  - name: "PodFitsPorts"
  - name: "MatchNodeSelector"
priorities:
...
  - name: "ZoneSpread"
    weight: 2
    argument:
      serviceAntiAffinity:
        label: "zone"
  - name: "ZonePreferred"
    weight: 1
    argument:
      labelPreference:
        label: "zone"
        presence: true
  - name: "ServiceSpreadingPriority"
    weight: 1
```

16.3. 再スケジュール (DESCHEDULING)

16.3.1. 概要

再スケジュール (Descheduling) には、Pod がより適切なノードに再スケジュールされるように 特定のポリシー に基づいて Pod をエビクトすることが関係しています。

クラスターは、さまざまな理由で、すでに実行中の Pod のスケジュールを解除および再スケジュールすることで恩恵を受けることができます。

- ノードの使用率が低くなっているか、または高くなっている。
- テイントまたはラベルなどの、Pod およびノードアフィニティーの各種要件が変更され、当初のスケジューリングの意思決定が特定のノードに適さなくなっている。
- ノードの障害により、Pod を移動する必要がある。
- 新規ノードがクラスターに追加されている。

Descheduler はエビクトされた Pod の置き換えをスケジュールしません。スケジューラー は、エビクトされた Pod に対してこのタスクを自動的に実行します。

DNS など、クラスターの完全な機能に欠かせないコアコンポーネントであるものの、マスターではなく通常のクラスターノードで実行されるコアコンポーネントが多数あることに留意する必要があります。クラスターはコンポーネントがエビクトされると正常な機能を停止する可能性があります。Descheduler がこれらの Pod を削除することを防ぐには、 **scheduler.alpha.kubernetes.io/critical-pod** アノテーションを Pod 仕様に追加して、Pod を Critical Pod として設定します。

注記

Descheduler ジョブは Critical Pod としてみなされ、これは Descheduler Pod が Descheduler のエビクトの対象になることを防ぎます。

Descheduler ジョブおよび Descheduler Pod は、デフォルトで作成される **kube-system** プロジェクトに作成されます。

重要

descheduler 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 の実稼働環境でのサービスレベルアグリーメント (SLA) ではサポートされていないため、Red Hat では実稼働環境での使用を推奨していません。テクノロジープレビューの機能は、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についての詳細は、https://access.redhat.com/support/offerings/techpreview/ を参照してください。

Descheduler は以下のタイプの Pod をエビクトしません。

- Critical Pod (**scheduler.alpha.kubernetes.io/critical-pod** アノテーションを持つ)。
- レプリカセット、レプリケーションコントローラー、デプロイメント、またはジョブに関連付けられていない Pod (静的およびミラー Pod またはスタンドアロンモードの Pod) (これらの Pod 再作成されないため)。
- DaemonSet に関連付けられた Pod。
- ローカルストレージを持つ Pod。
- Pod の Disruption Budget (PDB) が適用される Pod は、スケジュール解除が PDB に違反する場合にエビクトされません。Pod は、エビクションポリシー を使用してエビクトできます。

注記

Best Effort Pod は、Burstable および Guaranteed Pod の前にエビクトされます。

以下のセクションでは、Descheduler を設定し、実行するプロセスについて説明します。

1. ロールを作成します。
2. ポリシーファイル にスケジュール解除動作を定義します。
3. ポリシーファイルを参照するための参照マップ を作成します。
4. Descheduler ジョブ設定 を作成します。
5. Descheduler ジョブを実行します。

16.3.2. クラスターロールの作成

Descheduler が Pod で機能するために必要なパーミッションを設定するには、以下を実行します。

1. 以下のルールで クラスターロール を作成します。

```
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: descheduler-cluster-role
rules:
- apiGroups: [""]
  resources: ["nodes"]
```

```
    verbs: ["get", "watch", "list"] (1)
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "watch", "list", "delete"] (2)
- apiGroups: [""]
  resources: ["pods/eviction"] (3)
  verbs: ["create"]
```

(1) ノードの表示を許可するようにロールを設定します

(2) Pod の表示および削除を許可するようにロールを設定します。

(3) ノードがノードにバインドされた Pod をエビクトできるようにします。

2. ジョブを実行するために使用される サービスアカウント を作成します。

```
# oc create sa <file-name>.yaml -n kube-system
```

以下に例を示します。

```
# oc create sa descheduler-sa.yaml -n kube-system
```

3. クラスターロールをサービスアカウントにバインドします。

```
# oc create clusterrolebinding descheduler-cluster-role-binding \
    --clusterrole=<cluster-role-name> \
    --serviceaccount=kube-system:<service-account-name>
```

以下に例を示します。

```
# oc create clusterrolebinding descheduler-cluster-role-binding \
    --clusterrole=descheduler-cluster-role \
    --serviceaccount=kube-system:descheduler-sa
```

16.3.3. Descheduler ポリシーの作成

Descheduler は、YAML ポリシーファイルの **strategies** で定義されるルールに違反するノードから Pod を削除するように設定できます。次に、設定マップを使用して特定のスケジュール解除ストラテジーを適用するためにポリシーファイルへのパスと ジョブ仕様 が含まれる 設定マップ を作成します。

Descheduler ポリシーファイルのサンプル

```
apiVersion: "descheduler/v1alpha1"
kind: "DeschedulerPolicy"
strategies:
  "RemoveDuplicates":
     enabled: false
  "LowNodeUtilization":
     enabled: true
     params:
       nodeResourceUtilizationThresholds:
         thresholds:
```

```
          "cpu" : 20
          "memory": 20
          "pods": 20
        targetThresholds:
          "cpu" : 50
          "memory": 50
          "pods": 50
        numberOfNodes: 3
    "RemovePodsViolatingInterPodAntiAffinity":
      enabled: true
    "RemovePodsViolatingNodeAffinity":
      enabled: true
      params:
        nodeAffinityType:
        - "requiredDuringSchedulingIgnoredDuringExecution"
```

Descheduler で使用できるデフォルトストラテジーとして、3 つのストラテジーがあります。

- 重複 Pod の削除 (**RemoveDuplicates**)
- Pod の使用率の低いノードへの移動 (**LowNodeUtilization**)
- 非アフィニティールールに違反する Pod の削除 (**RemovePodsViolatingInterPodAntiAffinity**)
- ノードのアフィニティーに違反する Pod の削除 (RemovePodsViolatingNodeAffinity)

ストラテジーに関連付けられたパラメーターを必要に応じて設定し、無効にすることができます。

16.3.3.1. 重複 Pod の削除

RemoveDuplicates ストラテジーでは、1 つの Pod のみが同じノードで実行されている レプリカセット、レプリケーションコントローラー、デプロイメント設定、または ジョブ に関連付けられます。これらのオブジェクトに関連付けられている他の Pod がある場合、重複 Pod はエビクトされます。重複 Pod をエビクトすると、Pod をクラスター内により効果的に分散できます。

たとえば、ノードが失敗し、ノード上の Pod が別のノードに移行した場合に、複数の Pod が同じノードで実行されているレプリカセットまたはレプリケーションコントローラーに関連付けられると、重複 Pod が発生する可能性があります。失敗したノードが再び準備可能になると、それらの重複 Pod をエビクトするためにこのストラテジーが使用される可能性があります。

このストラテジーに関連付けられるパラメーターはありません。

```
apiVersion: "descheduler/v1alpha1"
kind: "DeschedulerPolicy"
strategies:
  "RemoveDuplicates":
     enabled: false (1)
```

(1) このポリシーを使用するには、この値を **enabled: true** に設定します。このポリシーを無効にするには、**false** に設定します。

16.3.3.2. ノードの低使用率ポリシー (Low Node Utilization Policy) の作成

LowNodeUtilization ストラテジーは、使用率の低いノードを検出し、他のノードから Pod をエビクトして、エビクトされた Pod がそれらの使用率の低いノードにスケジュールされるようにします。

ノードの使用率は、CPU、メモリーまたは Pod 数の設定可能なしきい値 **thresholds** (パーセンテージベース) で決定されます。ノードの使用率がこれらのしきい値のすべてを下回る場合、ノードの使用率は低いとみなされ、Descheduler は Pod を他のノードからエビクトする可能性があります。Pod の要求リソース要件は、ノードのリソース使用率を計算する際に考慮されます。

高いしきい値の **targetThresholds** は、使用率が適性なノードを判別するために使用されます。**thresholds** と **targetThresholds** 間にあるノードの使用率は適性であるとみなされ、エビクションの対象にはなりません。しきい値 **targetThresholds** は、CPU、メモリーおよび Pod 数について設定できます (パーセンテージベース)。

これらのしきい値はクラスター要件に合わせて調整できます。

numberOfNodes パラメーターは、使用率の低いノードの数が設定された値を上回る場合にのみストラテジーをアクティブにするために設定できます。いくつかのノードの使用率が低くなることが許容される場合にこのパラメーターを設定します。デフォルトで、**numberOfNodes** はゼロに設定されます。

```
apiVersion: "descheduler/v1alpha1"
kind: "DeschedulerPolicy"
strategies:
  "LowNodeUtilization":
    enabled: true
    params:
      nodeResourceUtilizationThresholds:
        thresholds: (1)
          "cpu" : 20
          "memory": 20
          "pods": 20
        targetThresholds: (2)
          "cpu" : 50
          "memory": 50
          "pods": 50
        numberOfNodes: 3 (3)
```

(1) ローエンドのしきい値を設定します。ノードがこれら 3 つの値のいずれよりも下回る場合、Descheduler はこのノードの使用率を低いとみなします。

(2) ハイエンドのしきい値を設定します。ノードがこれらの値を下回り、**threshold** 値を上回る場合、Descheduler はノードの使用率が適正であるとみなします。

(3) Descheduler が Pod を使用率の低いノードからエビクトする前に、使用率が低くなるノードの数を設定します。

16.3.3.3. Pod 間の非アフィニティー (Inter-Pod Anti-Affinity) に違反する Pod の削除

RemovePodsViolatingInterPodAntiAffinity ストラテジーは、Pod 間の非アフィニティー (inter-pod anti-affinity) に違反する Pod がノードから削除されるようにします。

たとえば、**Node1** には、**podA**、**podB**、および **podC** があります。**podB** および **podC** には非アフィニティールールがあり、これにより **podA** と同じノードでの実行が禁止されます。**podA** はノードからエビクトされ、**podB** および **podC** がそのノードで実行できるようになります。この状況は、**podB** および **podC** がノード上で実行されている際に、非アフィニティールールが適用される場合に生じます。

■

```
apiVersion: "descheduler/v1alpha1"
kind: "DeschedulerPolicy"
strategies:
  "RemovePodsViolatingInterPodAntiAffinity": (1)
    enabled: true
```

(1) このポリシーを使用するには、この値を **enabled: true** に設定します。このポリシーを無効にするには、**false** に設定します。

16.3.3.4. ノードアフィニティーに違反する Pod の削除

RemovePodsViolatingNodeAffinity ストラテジーにより、ノードアフィニティーに違反するすべての Pod がノードから確実に削除されます。この状態は、ノードが Pod のアフィニティールールを満たさなくなる場合に生じる可能性があります。アフィニティールールを満たす別のノードが利用可能な場合、Pod はエビクトされます。

たとえば、**podA** は、スケジューリング時に **requiredDuringSchedulingIgnoredDuringExecution** ノードアフィニティールールを満たしているために **nodeA** にスケジュールされます。**nodeA** がルールの条件を満たさなくなり、ノードアフィニティールールを満たす別のノードが利用可能な場合は、ストラテジーは **nodeA** から **podA** をエビクトし、これを他のノードに移動します。

```
apiVersion: "descheduler/v1alpha1"
kind: "DeschedulerPolicy"
strategies:
  "RemovePodsViolatingNodeAffinity": (1)
    enabled: true
    params:
      nodeAffinityType:
      - "requiredDuringSchedulingIgnoredDuringExecution" (2)
```

(1) このポリシーを使用するには、この値を **enabled: true** に設定します。このポリシーを無効にするには、**false** に設定します。

(2) **requiredDuringSchedulingIgnoredDuringExecution** ノードのアフィニティータイプを指定します。

16.3.4. Descheduler ポリシーの設定マップの作成

kube-system プロジェクトで Descheduler ポリシーファイルの 設定マップ を作成し、これが Descheduler ジョブで参照されるようにします。

```
# oc create configmap descheduler-policy-configmap \
    -n kube-system --from-file=<path-to-policy-dir/policy.yaml> (1)
```

(1) 作成したポリシーファイルのパスです。

16.3.5. ジョブ仕様の作成

Descheduler の ジョブ設定 を作成します。

```
apiVersion: batch/v1
```

```
kind: Job
metadata:
  name: descheduler-job
  namespace: kube-system
spec:
  parallelism: 1
  completions: 1
  template:
    metadata:
      name: descheduler-pod (1)
      annotations:
        scheduler.alpha.kubernetes.io/critical-pod: "true" (2)
    spec:
      containers:
      - name: descheduler
        image: registry.access.redhat.com/openshift3/ose-descheduler
        volumeMounts: (3)
        - mountPath: /policy-dir
          name: policy-volume
        command:
        - "/bin/sh"
        - "-ec"
        - |
          /bin/descheduler --policy-config-file /policy-dir/policy.yaml (4)
      restartPolicy: "Never"
      serviceAccountName: descheduler-sa (5)
      volumes:
      - name: policy-volume
        configMap:
          name: descheduler-policy-configmap
```

1. ジョブの名前を指定します。
2. スケジュール解除が実行されないように Pod を設定します。
3. ジョブがマウントされるコンテナーのボリューム名およびマウントパスです。
4. 作成した ポリシーファイル が保存されるコンテナー内のパスです。
5. 作成したサービスアカウントの名前を指定します。

ポリシーファイルは、設定マップからボリュームとしてマウントされます。

16.3.6. Descheduler の実行

Descheduler を Pod のジョブとして実行するには、以下を実行します。

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

以下に例を示します。

```
# oc create -f descheduler-job.yaml
```

16.4. カスタムスケジューリング

16.4.1. 概要

デフォルトのスケジューラーと共に複数のカスタムスケジューラーを実行し、各 Pod に使用できるスケジューラーを設定できます。

特定のスケジューラーを使用して指定された Pod をスケジュールするには、Pod 仕様にスケジューラーの名前を指定します。

注記

スケジューラーバイナリーの作成方法に関する情報は、本書では扱っておりません。たとえば、Kubernetes ドキュメントの Configure Multiple Schedulers を参照してください。

16.4.2. スケジューラーのパッケージ化

クラスターにカスタムスケジューラーを含む一般的なプロセスでは、イメージを作成し、そのイメージをデプロイメントに含める必要があります。

1. スケジューラーバイナリーをコンテナーイメージにパッケージ化します。
2. スケジューラーバイナリーを含む コンテナーイメージを作成します。
   以下に例を示します。

```
FROM <source-image>
ADD <path-to-binary> /usr/local/bin/kube-scheduler
```

3. ファイルを Dockerfile として保存し、イメージをビルドし、レジストリーにプッシュします。
   以下に例を示します。

```
docker build -t <dest_env_registry_ip>:<port>/<namespace>/<image name>:<tag>
docker push <dest_env_registry_ip>:<port>/<namespace>/<image name>:<tag>
```

4. OpenShift Container Platform で、カスタムスケジューラーのデプロイメントを作成します。

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: custom-scheduler
  namespace: kube-system
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: custom-scheduler
subjects:
- kind: ServiceAccount
  name: custom-scheduler
  namespace: kube-system
roleRef:
  kind: ClusterRole
```

```
  name: system:kube-scheduler
  apiGroup: rbac.authorization.k8s.io
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: custom-scheduler
  namespace: kube-system
  labels:
    app: custom-scheduler
spec:
  replicas: 1
  selector:
    matchLabels:
      app: custom-scheduler
  template:
    metadata:
      labels:
        app: custom-scheduler
    spec:
      serviceAccount: custom-scheduler
      containers:
        - name: custom-scheduler
          image: "<namespace>/<image name>:<tag>" (1)
          imagePullPolicy: Always
```

(1) カスタムスケジューラー用に作成したコンテナーイメージを指定します。

16.4.3. カスタムスケジューラーを使用した Pod のデプロイ

カスタムスケジューラーをクラスターにデプロイした後、デフォルトのスケジューラーではなくそのスケジューラーを使用するように Pod を設定できます。

1. Pod 設定を作成するか、または編集し、**schedulerName** パラメーターでスケジューラーの名前を指定します。名前は一意である必要があります。

スケジューラーを含む Pod 仕様のサンプル

```
apiVersion: v1
kind: Pod
metadata:
  name: custom-scheduler-example
  labels:
    name: custom-scheduler-example
spec:
  schedulerName: custom-scheduler (1)
  containers:
  - name: pod-with-second-annotation-container
    image: docker.io/ocpqe/hello-pod
```

(1) 使用するスケジューラーの名前です。スケジューラー名が指定されていない場合、Pod はデフォルトのスケジューラーを使用して自動的にスケジュールされます。

2. 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 custom-scheduler-example.yaml
```

3. 以下のコマンドを実行して、Pod が作成されていることを確認します。

```
$ oc get pod <file-name>
```

以下に例を示します。

```
$ oc get pod custom-scheduler-example

NAME                      READY     STATUS    RESTARTS   AGE
custom-scheduler-example   1/1       Running   0          4m
```

4. 以下のコマンドを実行して、カスタムスケジューラーが Pod をスケジュールしていることを確認します。

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

以下に例を示します。

```
$ oc describe pod custom-scheduler-example
```

以下の切り捨てられた出力に示されるように、スケジューラーの名前が一覧表示されます。

```
...

Events:
  FirstSeen  LastSeen  Count  From              SubObjectPath  Type       Reason Message
  ---------  --------  -----  ----              -------------  --------   ------ -------
  1m         1m        1      custom-scheduler    Normal         Scheduled  Successfully assigned
custom-scheduler to <$node1>

...
```

16.5. POD 配置の制御

16.5.1. 概要

クラスター管理者は、特定のロールを持つアプリケーション開発者が Pod のスケジュール時に特定ノードをターゲットとすることを防ぐポリシーを設定できます。

Pod ノード制約の受付コントローラーは、Pod がラベルを使用して指定されたノードホストのみにデプロイされるようにし、特定のロールを持たないユーザーが **nodeSelector** フィールドを使用して Pod をスケジュールできないようにします。

16.5.2. ノード名の使用による Pod 配置の制約

Pod ノード制約の受付コントローラーを使用し、Pod にラベルを割り当て、これを Pod 設定の **nodeName** 設定に指定することで、Pod が指定されたノードホストにのみデプロイされるようにします。

1. 必要なラベル (詳細は、ノードでのラベルの更新 を参照) および ノードセレクター が環境にセットアップされていることを確認します。
   たとえば、Pod 設定が必要なラベルを示す **nodeName** 値を持つことを確認します。

```
apiVersion: v1
kind: Pod
spec:
  nodeName: <value>
```

2. マスター設定ファイル (**/etc/origin/master/master-config.yaml**) を変更して **PodNodeConstraints** を **admissionConfig** セクションに追加します。

```
...
admissionConfig:
  pluginConfig:
    PodNodeConstraints:
      configuration:
        apiversion: v1
        kind: PodNodeConstraintsConfig
...
```

3. 変更を有効にするために OpenShift Container Platform を再起動します。

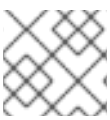
```
# master-restart api
# master-restart controllers
```

16.5.3. ノードセレクターの使用による Pod 配置の制約

ノードセレクター を使用して、Pod が特定のラベルを持つノードにのみ配置されるようにすることができます。クラスター管理者は、Pod ノード制約の受付コントローラーを使用して、**pods/binding** パーミッションのないユーザーがノードセレクターを使用して Pod をスケジュールできないようにするポリシーを設定できます。

マスター設定ファイルの **nodeSelectorLabelBlacklist** フィールドを使用して、一部のロールが Pod 設定の **nodeSelector** フィールドで指定できるラベルを制御できます。**pods/binding** パーミッション ロール を持つユーザー、サービスアカウントおよびグループは任意のノードセレクターを指定できます。**pods/binding** パーミッションがない場合は、**nodeSelectorLabelBlacklist** に表示されるすべてのラベルに **nodeSelector** を設定することは禁止されます。

たとえば、OpenShift Container Platform クラスターは、2 つの地域にまたがる 5 つのデータセンターで設定される場合があります。米国の **us-east**、**us-central**、および **us-west**、およびアジア太平洋 (APAC) の **apac-east** および **apac-west** です。それぞれの地理的地域の各ノードには、それらに応じたラベルが付けられます。たとえば、**region: us-east** のようになります。



注記

ラベルの割り当ての詳細は、ノードでのラベルの更新 を参照してください。

クラスター管理者は、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。ノードセレクターを作成し、米国のデータセンターを **superregion: us** に、APAC のデータセンターを **superregion: apac** に分類できます。

データセンターごとのリソースの均等なロードを維持するには、必要な **region** をマスター設定の **nodeSelectorLabelBlacklist** セクションに追加できます。その後は、米国の開発者が Pod を作成するたびに、Pod は **superregion: us** ラベルの付いた地域のいずれかにあるノードにデプロイされます。開発者が Pod に特定の region (地域) をターゲットに設定しようとすると (例: **region: us-east**)、エラーが出されます。これを Pod にノードセレクターを設定せずに試行すると、ターゲットとした region (地域) にデプロイすることができます。 それは **superregion: us** がプロジェクトレベルのノードセレクターとして設定されており、**region: us-east** というラベルが付けられたノードには **superregion: us** というラベルも付けられているためです。

1. 必要なラベル (詳細は、ノードでのラベルの更新 を参照) および ノードセレクター が環境にセットアップされていることを確認します。
   たとえば、Pod 設定が必要なラベルを示す **nodeSelector** 値を持つことを確認します。

```
apiVersion: v1
kind: Pod
spec:
  nodeSelector:
    <key>: <value>
...
```

2. マスター設定ファイル **/etc/origin/master/master-config.yaml** を変更し、**nodeSelectorLabelBlacklist** を、Pod の配置を拒否する必要のあるノードホストに割り当てられるラベルと共に **admissionConfig** セクションに追加します。

```
...
admissionConfig:
  pluginConfig:
    PodNodeConstraints:
      configuration:
        apiversion: v1
        kind: PodNodeConstraintsConfig
        nodeSelectorLabelBlacklist:
          - kubernetes.io/hostname
          - <label>
...
```

3. 変更を有効にするために OpenShift Container Platform を再起動します。

```
# master-restart api
# master-restart controllers
```

16.5.4. プロジェクトへの Pod 配置の制御

Pod ノードセレクターの受付コントローラーを使用して、Pod を特定のプロジェクトに関連付けられたノードに対して強制的に適用したり、Pod がそれらのノードでスケジュールされないようにしたりできます。

Pod ノードセレクターの受付コントローラーは、プロジェクトのラベル と Pod で指定されるノードセレクターを使用して Pod を配置する場所を決定します。新規 Pod は、Pod のノードセレクターがプロジェクトのラベルに一致する場合にのみプロジェクトに関連付けられたノードに配置されます。

Pod の作成後に、ノードセレクターは Pod にマージされ、Pod 仕様に元々含まれていたラベルとノードセレクターの新規ラベルが含まれるようにします。以下の例は、マージの結果について示しています。

Pod ノードセレクターの受付コントローラーにより、特定のプロジェクトで許可されるラベルの一覧を作成することもできます。この一覧は開発者がプロジェクトで使用できるラベルを認識するための ホワイトリスト として機能し、管理者がクラスターでのラベル設定の制御を強化するのに役立ちます。

Pod ノードセレクター の受付コントローラーをアクティブにするには、以下を実行します。

1. 以下の方法のいずれかを使用して **Pod** ノードセレクター の受付コントローラーとホワイトリストを設定します。

   - 以下をマスター設定ファイル (**/etc/origin/master/master-config.yaml**) に追加します。

```
admissionConfig:
  pluginConfig:
    PodNodeSelector:
      configuration:
        podNodeSelectorPluginConfig: 1
          clusterDefaultNodeSelector: "k3=v3" 2
          ns1: region=west,env=test,infra=fedora,os=fedora 3
```

   1 **Pod** ノードセレクター の受付コントローラープラグインを追加します。

   2 すべてのノードのデフォルトラベルを作成します。

   3 指定されたプロジェクトで許可されるラベルのホワイトリストを作成します。ここで、プロジェクトは **ns1** で、ラベルはそれに続く **key=value** ペアになります。

   - 受付コントローラーの情報を含むファイルを作成します。

```
podNodeSelectorPluginConfig:
   clusterDefaultNodeSelector: "k3=v3"
    ns1: region=west,env=test,infra=fedora,os=fedora
```

   次に、マスター設定でファイルを参照します。

```
admissionConfig:
  pluginConfig:
    PodNodeSelector:
      location: <path-to-file>
```

注記

プロジェクトにノードセレクターが指定されていない場合、そのプロジェクトに関連付けられた Pod はデフォルトのノードセレクター (**clusterDefaultNodeSelector**) を使用してマージされます。

2. 変更を有効にするために OpenShift Container Platform を再起動します。

```
# master-restart api
# master-restart controllers
```

3. **scheduler.alpha.kubernetes.io/node-selector** アノテーションおよびラベルを含むプロジェクトオブジェクトを作成します。

```
apiVersion: v1
kind: Namespace
metadata
  name: ns1
  annotations:
    scheduler.alpha.kubernetes.io/node-selector: env=test,infra=fedora (1)
spec: {},
status: {}
```

(1) プロジェクトのラベルセレクターに一致するラベルを作成するためのアノテーションです。ここで、キー/値のラベルは **env=test** および **infra=fedora** になります。

注記

Pod Node Selector 受付コントローラーを使用している場合、プロジェクトノードセレクターを設定するるために **oc adm new-project <project-name>** を設定することはできません。**oc adm new-project myproject --node-selector='type=user-node,region=<region>** コマンドを使用してプロジェクトノードセレクターを設定する場合、OpenShift Container Platform は **openshift.io/node-selector** アノテーションを設定します。 これは **NodeEnv** 受付プラグインで処理されます。

4. ノードセレクターにラベルを含む Pod 仕様を作成します。以下は例になります。

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    name: hello-pod
  name: hello-pod
spec:
  containers:
    - image: "docker.io/ocpqe/hello-pod:latest"
      imagePullPolicy: IfNotPresent
      name: hello-pod
      ports:
        - containerPort: 8080
          protocol: TCP
      resources: {}
      securityContext:
        capabilities: {}
        privileged: false
      terminationMessagePath: /dev/termination-log
  dnsPolicy: ClusterFirst
  restartPolicy: Always
  nodeSelector: (1)
    env: test
    os: fedora
  serviceAccount: ""
status: {}
```

1 プロジェクトラベルに一致するノードセレクター。

5. プロジェクトに Pod を作成します。

```
# oc create -f pod.yaml --namespace=ns1
```

6. ノードセレクターのラベルが Pod 設定に追加されていることを確認します。

```
get pod pod1 --namespace=ns1 -o json

nodeSelector": {
 "env": "test",
 "infra": "fedora",
 "os": "fedora"
}
```

ノードセレクターは Pod にマージされ、Pod は適切なプロジェクトでスケジュールされます。

プロジェクト仕様で指定されていないラベルを使って Pod を作成する場合、Pod はノードでスケジュールされません。

たとえば、ここでラベル **env: production** は、いずれのプロジェクト仕様にも含まれていません。

```
nodeSelector:
 "env: production"
 "infra": "fedora",
 "os": "fedora"
```

ノードセレクターのアノテーションのないノードがある場合は、Pod はそこにスケジュールされます。

16.6. POD の優先順位とプリエンプション

16.6.1. Pod の優先順位およびプリエンプションの適用

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 の優先順位は、他の Pod との対比で Pod の重要度を示し、その優先順位に基づいて 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 をスケジュールできます。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。

優先順位およびプリエンプションを使用するには、Pod の相対的な重みを定義する優先順位クラスを作成します。次に Pod 仕様で優先順位クラスを参照し、スケジューリングの重みを適用します。

プリエンプションはスケジューラー設定ファイルの **disablePreemption** パラメーターで制御されます。 これはデフォルトで **false** に設定されます。

16.6.2. 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が有効にされる場合、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

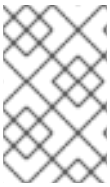
を満たす場合に優先順位の低い Pod よりも早くスケジュールされる可能性があります。Pod をスケジュールできない場合、スケジューラーは引き続き他の優先順位の低い Pod をスケジュールします。

16.6.2.1. 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 億より大きい数を予約します。デフォルトで、OpenShift Container Platform には 2 つの予約された優先順位クラスがあり、これらは重要なシステム Pod で保証されたスケジューリングが適用されるために使用されます。

- **System-node-critical**: この優先順位クラスには 2000001000 の値があり、ノードからエビクト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。
- **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。



注記

既存クラスターをアップグレードする場合、既存 Pod の優先順位はゼロになります。ただし、**scheduler.alpha.kubernetes.io/critical-pod** アノテーションを持つ既存 Pod は **system-cluster-critical** クラスに自動的に変換されます。

16.6.2.2. 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。

以下の YAML は、前述の例で作成された優先順位クラスを使用する Pod 設定の例です。優先順位の受付コントローラーは仕様をチェックし、Pod の優先順位を 1000000 に解決します。

16.6.3. Pod プリエンプションについて

開発者が Pod を作成する場合、Pod はキューに入れられます。Pod の優先順位およびプリエンプション機能が有効にされている場合、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、プリエンプ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 のプリエンプ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プ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 のプリエンプ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 のプリエンプションを実行し、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。

プリエンプションは、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 のプリエンプションを実行するノードを考慮します。

16.6.3.1. Pod プリエンプ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プ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、OpenShift Container Platform は、 Best Effort レベルで Pod のプリエンプ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 のプリエンプ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 のプリエンプ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 のプリエンプ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。

16.6.3.2. プリエンプションが実行された Pod の正常な終了

Pod のプリエンプションの実行中、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 のプリエンプ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。

16.6.4. Pod の優先順位のシナリオ例

Pod の優先順位およびプリエンプションはスケジューリングに使用する優先順位を Pod に割り当てます。スケジューラーは優先順位の高い Pod をスケジュールするために優先順位の低い Pod のプリエンプション (エビクト) を実行します。

通常のプリエンプションシナリオ

Pod P は保留中の Pod です。

1. スケジューラーは ノード **N** を見つけます。 ここでは、1 つ以上の Pod が削除され、 **Pod P** がそのノードにスケジュールされます。
2. スケジューラーは ノード **N** から優先順位の低い Pod を削除し、 **Pod P** をこのノードにスケジュールします。
3. **Pod P** の **nominatedNodeName** フィールドは、 ノード **N** の名前に設定されます。

注記

Pod P は必ずしも指定したノードにスケジュールされる訳ではありません。

プリエンプションおよび終了期間

プリエンプションが実行された Pod には長い終了期間が設定されます。

1. スケジューラーは ノード **N** で優先順位の低い Pod のプリエンプションを実行します。
2. スケジューラーは、Pod が正常に終了するのを待機します。
3. 他のスケジューリングの理由により、ノード **M** が利用可能になります。
4. スケジューラーは **Pod P** を ノード **M** にスケジュールできます。

16.6.5. 優先順位およびプリエンプションの設定

priorityClassName を Pod 仕様で使用し、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 の優先順位およびプリエンプションを適用できます。

優先順位クラスオブジェクトのサンプル

```
apiVersion: scheduling.k8s.io/v1beta1
kind: PriorityClass
metadata:
  name: high-priority (1)
value: 1000000 (2)
globalDefault: false (3)
description: "This priority class should be used for XYZ service pods only." (4)
```

(1) 優先順位クラスオブジェクトの名前です。

(2) オブジェクトの優先順位の値です。

(3) この優先順位クラスが優先順位クラス名が指定されない状態で Pod に使用されるかどうかを示すオプションのフィールドです。このフィールドはデフォルトで **false** です。**globalDefault** が **true** に設定される 1 つの優先順位クラスのみがクラスター内に存在できます。**globalDefault:true** が設定された優先順位クラスがない場合、優先順位クラス名が設定されていない Pod の優先順位はゼロになります。**globalDefault:true** が設定された優先順位クラスを追加すると、優先順位クラスが追加された後に作成された Pod のみがその影響を受け、これによって既存 Pod の優先順位は変更されません。

4 開発者がこの優先順位クラスで使用する必要のある Pod を記述するオプションのテキスト文字列です。

優先順位クラス名を持つ Pod 仕様サンプル

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    env: test
spec:
  containers:
  - name: nginx
    image: nginx
    imagePullPolicy: IfNotPresent
  priorityClassName: high-priority 1
```

1 この Pod で使用する優先順位クラスを指定します。

優先順位およびプリエンプ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. 1 つ以上の優先順位クラスを作成します。
   a. 優先順位の名前および値を指定します。
   b. 優先順位クラスおよび説明に **globalDefault** フィールドをオプションで指定します。
2. Pod を作成するか、または優先順位クラスの名前を含むように既存 Pod を編集します。優先順位名は Pod 設定または Pod テンプレートに直接追加できます。

16.6.6. 優先順位およびプリエンプションの無効化

Pod の優先順位およびプリエンプション機能を無効にできます。

この機能が無効にされた後に、既存 Pod はそれらの優先順位フィールドを維持しますが、プリエンプションは無効にされ、優先順位フィールドは無視されます。この機能が無効にされると、新規 Pod に優先順位クラス名を設定できません。

重要

クラスターがリソース不足の状態にある場合、Critical Pod のスケジューリングにはスケジューラーのプリエンプションが使用されます。このため、プリエンプションを無効にしないことが推奨されています。DaemonSet Pod は DaemonSet コントローラーによってスケジュールされるため、プリエンプションを無効にしても影響を受けません。

クラスターのプリエンプションを無効にするには、以下を実行します。

1. **schedulerArgs** セクションの **disablePreemption** パラメーターを **false** に設定するように **master-config.yaml** を変更します。

```
disablePreemption=false
```

2. 変更を有効にするために OpenShift Container Platform マスターサービスおよびスケジューラーを再起動します。

```
# master-restart api
# master-restart scheduler
```

16.7. 詳細スケジューリング

16.7.1. 概要

詳細スケジューリングには、Pod が特定ノードで実行されることを要求したり、Pod が特定ノードで実行されることが優先されるように Pod を設定することが関係します。

OpenShift Container Platform は Pod を適切な方法で自動的に配置するため、通常、詳細スケジューリングは必要ありません。たとえば、デフォルトスケジューラーは Pod をノード間で均等に分散し、ノードの利用可能なリソースを考慮します。ただし、Pod を配置する場所についてはさらに制御を強化することをお勧めします。

Pod をより高速なディスクが搭載されたマシンに配置する必要ある場合 (またはそのマシンに配置するのを防ぐ場合)、または 2 つの異なるサービスの Pod が相互に通信できるように配置する必要がある場合、詳細スケジューリングを使用してそれを可能にすることができます。

適切な新規 Pod を特定のノードグループにスケジュールし、その他の新規 Pod がそれらのノードでスケジュールされるのを防ぐには、必要に応じてこれらの方法を組み合わせることができます。

16.7.2. 詳細スケジューリングの使用

クラスターで詳細スケジューリングを起動する方法はいくつかあります。

Pod のアフィニティーおよび非アフィニティー

Pod のアフィニティーにより、**Pod** がその配置に使用できるアフィニティー (または非アフィニティー) を、(セキュリティー上の理由によるアプリケーションの待機時間の要件などのために) **Pod** のグループに対して指定できるようにします。ノード自体は配置に対して制御を行いません。
Pod のアフィニティーはノードのラベルと Pod のラベルセレクターを使用して Pod 配置のルールを作成します。ルールは mandatory (必須) または best-effort (優先) のいずれかにすることができます。

Pod のアフィニティーおよび非アフィニティーの使用 を参照してください。

ノードのアフィニティー

ノードのアフィニティーにより、**Pod** がその配置に使用できるアフィニティー (または非アフィニティー) を、(高可用性のための特殊なハードウェア、場所、要件などにより) ノード のグループに対して指定できるようにします。ノード自体は配置に対して制御を行いません。
ノードのアフィニティーはノードのラベルと Pod のラベルセレクターを使用して Pod 配置のルールを作成します。ルールは mandatory (必須) または best-effort (優先) のいずれかにすることができます。

ノードアフィニティーの使用 を参照してください。

ノードセレクター

ノードセレクターは詳細スケジューリングの最も単純な形態です。ノードのアフィニティーのように、ノードセレクターはノードのラベルと Pod のラベルセレクターを使用し、**Pod** がその配置に使

用する ノード を制御できるようにします。ただし、ノードセレクターにはノードのアフィニティーが持つ required (必須) ルールまたは preferred (優先) ルールはありません。
ノードセレクターの使用 を参照してください。

テイントおよび容認 (Toleration)

テイント/容認により、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。テイントはノードのラベルであり、容認は Pod のラベルです。スケジュールを可能にするには、Pod のラベルは ノードのラベル (テイント) に一致する (またはこれを許容する) 必要があります。
テイント/容認にはアフィニティーと比較して 1 つ利点があります。たとえばアフィニティーの場合は、異なるラベルを持つノードの新規グループをクラスターに追加する場合、ノードにアクセスさせたい Pod と新規ノードを使用させたくない Pod のそれぞれに対してアフィニティーを更新する必要があります。テイント/容認の場合には、新規ノードに到達させる必要のある Pod のみを更新すれば、他の Pod は拒否されることになります。

テイントおよび容認の使用 を参照してください。

16.8. 詳細スケジューリングおよびノードのアフィニティー

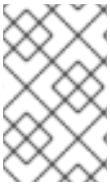
16.8.1. 概要

ノードのアフィニティー は、Pod の配置場所を判別するためにスケジューラーによって使用されるルールのセットです。ルールはカスタムの ノードのラベル と Pod で指定されるラベルセレクターを使って定義されます。ノードのアフィニティーにより、**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ー (または非アフィニティー) 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ー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。

16.8.2. ノードのアフィニティーの設定

ノードのアフィニティーは、Pod 仕様で設定することができます。required (必須) ルール、preferred (優先) ルール のいずれかまたは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:
  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
```

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:
  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
```

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。

16.8.2.1. 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。

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. ノード設定を編集するか、または **oc label node** コマンドを使用して、ラベルをノードに追加します。

```
$ oc label node node1 e2e-az-name=e2e-az1
```

注記

クラスターのノードを変更するには、ノード設定マップ を必要に応じて更新します。**node-config.yaml** ファイルは手動で変更しないようにしてください。

2. Pod 仕様では、**nodeAffinity** スタンザを使用して **requiredDuringSchedulingIgnoredDuringExecution** パラメーターを設定します。

   a. 満たしている必要のあるキーおよび値を指定します。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 **value** パラメーターを使用します。

   b. **operator** を指定します。演算子は **In**、**NotIn**、**Exists**、**DoesNotExist**、**Lt**、または **Gt** にすることができます。たとえば、演算子 **In** を使用して、ラベルがノードにあることを要求します。

```
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: e2e-az-name
            operator: In
            values:
            - e2e-az1
            - e2e-az2
```

3. Pod を作成します。

```
$ oc create -f e2e-az2.yaml
```

16.8.2.2. 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。

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. ノード設定を編集するか、または **oc label node** コマンドを実行して、ラベルをノードに追加します。

```
$ oc label node node1 e2e-az-name=e2e-az3
```

注記

クラスターのノードを変更するには、ノード設定マップ を必要に応じて更新します。**node-config.yaml** ファイルは手動で変更しないようにしてください。

2. Pod 仕様では、**nodeAffinity** スタンザを使用して **preferredDuringSchedulingIgnoredDuringExecution** パラメーターを設定します。

   a. 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。

   b. 満たしている必要のあるキーおよび値を指定します。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 **value** パラメーターを使用します。

```
    preferredDuringSchedulingIgnoredDuringExecution:
    - weight: 1
      preference:
        matchExpressions:
        - key: e2e-az-name
          operator: In
          values:
          - e2e-az3
```

3. **operator** を指定します。演算子は **In**、**NotIn**、**Exists**、**DoesNotExist**、**Lt**、または **Gt** 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。

4. Pod を作成します。

```
$ oc create -f e2e-az3.yaml
```

16.8.3. 例

以下の例は、ノードのアフィニティーを示しています。

16.8.3.1. 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
```

- Pod **pod-s1** 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
apiVersion: v1
kind: Pod
metadata:
  name: pod-s1
spec:
  containers:
    - image: "docker.io/ocpqe/hello-pod"
      name: hello-pod
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
          - matchExpressions:
            - key: "zone"
              operator: In
              values:
              - us
```

- 標準コマンドを使用して Pod を作成します。

```
$ oc create -f pod-s1.yaml
pod "pod-s1" created
```

- Pod **pod-s1** を **Node1** にスケジュールできます。

```
$ oc get pod -o wide
NAME     READY     STATUS      RESTARTS   AGE      IP      NODE
pod-s1   1/1       Running     0          4m       IP1     node1
```

16.8.3.2. 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
```

- Pod **pod-s1** 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
apiVersion: v1
kind: Pod
metadata:
  name: pod-s1
spec:
  containers:
    - image: "docker.io/ocpqe/hello-pod"
      name: hello-pod
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
          - matchExpressions:
            - key: "zone"
              operator: In
              values:
              - us
```

- Pod **pod-s1** は **Node1** にスケジュールすることができません。

```
$ oc describe pod pod-s1


...
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).
```

16.9. 詳細スケジューリングおよび POD のアフィニティーと非アフィニティー

16.9.1. 概要

Pod のアフィニティー および **Pod** の非アフィニティー により、他の Pod との関連で Pod を配置する方法についてのルールを指定できます。ルールはカスタムの ノードのラベル と Pod で指定されるラベルセレクターを使って定義されます。Pod のアフィニティー/非アフィニティーにより、**Pod** はアフィニティー (または非アフィニティー) を、その配置に使用できる **Pod** のグループに対して指定できます。ノード自体は配置に対して制御を行いません。

たとえば、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またはアベイラビリティーゾーン間でサービスの Pod を分散することもできます。

Pod のアフィニティー/非アフィニティーにより、他の Pod のラベルに基づいて Pod のスケジュール対象とするノードを制限することができます。ラベル はキー/値のペアです。

- Pod のアフィニティーはスケジューラーに対し、新規 Pod のラベルセレクターが現在の 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 のラベルセレクターが現在の 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。

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ら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。

16.9.2. Pod のアフィニティーおよび非アフィニティーの設定

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 のいずれかまたは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 がすでに実行されている場合にのみ Pod をノードにスケジュール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 がすでに実行されている場合は Pod をノードにスケジュール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 設定のサンプル

```
apiVersion: v1
kind: Pod
metadata:
```

```
  name: with-pod-affinity
spec:
  affinity:
    podAffinity: 1
      requiredDuringSchedulingIgnoredDuringExecution: 2
      - labelSelector:
          matchExpressions:
          - key: security 3
            operator: In 4
            values:
            - S1 5
        topologyKey: failure-domain.beta.kubernetes.io/zone
  containers:
  - name: with-pod-affinity
    image: docker.io/ocpqe/hello-pod
```

1 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 設定のサンプル

```
apiVersion: v1
kind: Pod
metadata:
  name: with-pod-antiaffinity
spec:
  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
```

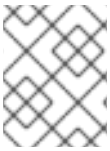
1 Pod の非アフィニティーを設定するためのスタンザです。

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。



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す。

16.9.2.1. 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。

1. Pod 仕様の特定のラベルの付いた Pod を作成します。

```
$ cat team4.yaml
apiVersion: v1
kind: Pod
metadata:
  name: security-s1
  labels:
    security: S1
spec:
  containers:
  - name: security-s1
    image: docker.io/ocpqe/hello-pod
```

2. 他の Pod の作成時に、以下のように Pod 仕様を編集します。

   a. **podAffinity** スタンザを使用して、**requiredDuringSchedulingIgnoredDuringExecution** パラメーターまたは **preferredDuringSchedulingIgnoredDuringExecution** パラメーターを設定します。

   b. 満たしている必要のあるキーおよび値を指定します。新規 Pod を他の Pod と共にスケジュールする必要がある場合、最初の Pod のラベルと同じ **key** および **value** パラメーターを使用します。

```
podAffinity:
  requiredDuringSchedulingIgnoredDuringExecution:
  - labelSelector:
      matchExpressions:
      - key: security
        operator: In
        values:
        - S1
    topologyKey: failure-domain.beta.kubernetes.io/zone
```

c. **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。

d. **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。

3. Pod を作成します。

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

16.9.2.2. 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 のスケジュール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。

1. Pod 仕様の特定のラベルの付いた Pod を作成します。

```
$ cat team4.yaml
apiVersion: v1
kind: Pod
metadata:
  name: security-s2
  labels:
    security: S2
spec:
  containers:
  - name: security-s2
    image: docker.io/ocpqe/hello-pod
```

2. 他の Pod の作成時に、Pod 仕様を編集して以下のパラメーターを設定します。

3. **podAffinity** スタンザを使用して、**requiredDuringSchedulingIgnoredDuringExecution** パラメーターまたは **preferredDuringSchedulingIgnoredDuringExecution** パラメーターを設定します。

a. 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。

b. 満たしている必要のあるキーおよび値を指定します。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 **value** パラメーターを使用します。

```
podAntiAffinity:
  preferredDuringSchedulingIgnoredDuringExecution:
  - weight: 100
    podAffinityTerm:
      labelSelector:
        matchExpressions:
        - key: security
          operator: In
          values:
          - S2
      topologyKey: kubernetes.io/hostname
```

c. preferred (優先) ルールの場合、重みを 1-100 で指定します。

d. **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。

4. **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。

5. Pod を作成します。

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

16.9.3. 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。

16.9.3.1. 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。

```
$ cat team4.yaml
apiVersion: v1
kind: Pod
metadata:
  name: team4
  labels:
    team: "4"
spec:
  containers:
  - name: ocp
    image: docker.io/ocpqe/hello-pod
```

- 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。

```
$ cat pod-team4a.yaml
apiVersion: v1
kind: Pod
metadata:
  name: team4a
spec:
  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
```

- **team4a** Pod は **team4** Pod と同じノードにスケジュールされます。

16.9.3.2. 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。

```
$ cat pod-s1.yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod-s1
  labels:
    security: s1
spec:
  containers:
  - name: ocp
    image: docker.io/ocpqe/hello-pod
```

- 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。

```
$ cat pod-s2.yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod-s2
spec:
  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
```

- Pod **pod-s2** は **pod-s1** と同じノードにスケジュールできません。

16.9.3.3. 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。

```
$ cat pod-s1.yaml
apiVersion: v1
```

```
kind: Pod
metadata:
  name: pod-s1
  labels:
    security: s1
spec:
  containers:
  - name: ocp
    image: docker.io/ocpqe/hello-pod
```

- Pod **pod-s2** にはラベルセレクター **security:s2** があります。

```
$ cat pod-s2.yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod-s2
spec:
  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
```

- 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。

```
NAME     READY     STATUS    RESTARTS   AGE       IP        NODE
pod-s2   0/1       Pending   0          32s       <none>
```

16.10. 詳細スケジューリングおよびノードセレクター

16.10.1. 概要

ノードセレクター はキーと値のペアのマップを指定します。ルールは、カスタムの ノードのラベル および Pod で指定されるセレ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 ノードセレクター を使用している場合は、以下の 重要な考慮事項 を参照してください。

16.10.2. ノードセレクターの設定

Pod 設定で **nodeSelector** を使用することで、Pod を特定のラベルの付いたノードのみに配置することができます。

1. 必要なラベル (詳細は、ノードでのラベルの更新 を参照) および ノードセレクター が環境にセットアップされていることを確認します。
   たとえば、Pod 設定が必要なラベルを示す **nodeSelector** 値を持つことを確認します。

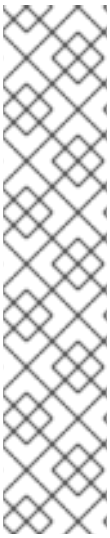
```
apiVersion: v1
kind: Pod
spec:
  nodeSelector:
    <key>: <value>
...
```

2. マスター設定ファイル **/etc/origin/master/master-config.yaml** を変更し、**nodeSelectorLabelBlacklist** を、Pod の配置を拒否する必要のあるノードホストに割り当てられるラベルと共に **admissionConfig** セクションに追加します。

```
...
admissionConfig:
  pluginConfig:
    PodNodeConstraints:
      configuration:
        apiversion: v1
        kind: PodNodeConstraintsConfig
        nodeSelectorLabelBlacklist:
          - kubernetes.io/hostname
          - <label>
...
```

3. 変更を有効にするために OpenShift Container Platform を再起動します。

```
# master-restart api
# master-restart controllers
```



注記

同じ Pod 設定でノードセレクターと 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。

16.11. 詳細スケジューリングおよび容認

16.11.1. 概要

テイントおよび容認により、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。

16.11.2. テイントおよび容認 (Toleration)

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。

テイントはノード仕様 (**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。ノードのテイントはノードに対し、テイントを容認しないすべての Pod を拒否するよう指示します。

テイントおよび容認は、key、value、および effect.で設定されています。演算子により、これらの 3 つのパラメーターのいずれかを空のままにすることができます。

表16.1 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。 **NoSchedule** • 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** • 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が、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** • 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。

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。

16.11.2.1. 複数テイントの使用

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。OpenShift Container Platform 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。
   - effect が **NoSchedule** の一致しないテイントが 1 つ以上ある場合、OpenShift Container Platform は Pod をノードにスケジュールできません。
   - effect が **NoSchedule** の一致しないテイントがなく、effect が **PreferNoSchedule** の一致しない テイントが 1 つ以上ある場合、OpenShift Container Platform は Pod のノードへのスケジュールを試行しません。
   - effect が **NoExecute** のテイントが 1 つ以上ある場合、OpenShift Container Platform は Pod をノードからエビクトするか (ノードですでに実行中の場合)、または Pod のそのノードへのスケジュールが実行されません (ノードでまだ実行されていない場合)。
     - テイントを容認しない Pod はすぐにエビクトされます。
     - 容認の仕様に **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。

- ノードには以下のテイントがあります。

```
$ 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。

```
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。

16.11.3. テイントの既存ノードへの追加

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 **oc adm taint** 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
```

この例では、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。

16.11.4. 容認の Pod への追加

容認を Pod に追加するには、Pod 仕様を **tolerations** セクションを含めるように編集します。

Equal 演算子を含む Pod 設定ファイルのサンプル

```
tolerations:
- key: "key1" 1
  operator: "Equal" 2
  value: "value1" 3
  effect: "NoExecute" 4
  tolerationSeconds: 3600 5
```

1 2 3 4 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。

5 **tolerationSeconds** パラメーターは、Pod がエビクトされる前にノードにバインドされる期間を指定します。以下の Pod エビクションを遅延させる容認期間 (秒数) の使用 を参照してください。

Exists 演算子を含む Pod 設定ファイルのサンプル

```
tolerations:
- key: "key1"
  operator: "Exists"
  effect: "NoExecute"
  tolerationSeconds: 3600
```

これらの容認のいずれも 上記の **oc adm taint** コマンドで作成されるテイント に一致します。いずれかの容認のある Pod は **node1** にスケジュールできます。

16.11.4.1. Pod のエビクションを遅延させる容認期間 (秒数) の使用

Pod 仕様に **tolerationSeconds** パラメーターを指定して、Pod がエビクト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 **NoExecute** のあるテイントがノードに追加される場合、テイントを容認しない Pod は即時にエビクトされます (テイントを容認する Pod はエビクトされません)。ただし、エビクトされる Pod に **tolerationSeconds** パラメーターがある場合、Pod は期間切れになるまでエビクトされません。

以下に例を示します。

```
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、一致するテイントがない場合、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。

16.11.4.1.1. 容認の秒数のデフォルト値の設定

このプラグインは、**node.kubernetes.io/not-ready:NoExecute** および **node.kubernetes.io/unreachable:NoExecute** テイントを 5 分間容認するための Pod のデフォルトの容認を設定します。

ユーザーが提供する Pod 設定にいずれかの容認がある場合、デフォルトは追加されません。

デフォルトの容認の秒数を有効にするには、以下を実行します。

1. マスター設定ファイル (**/etc/origin/master/master-config.yaml**) を変更して、**DefaultTolerationSeconds** を admissionConfig セクションに追加します。

```
admissionConfig:
  pluginConfig:
    DefaultTolerationSeconds:
      configuration:
```

```
    kind: DefaultAdmissionConfig
    apiVersion: v1
    disable: false
```

2. 変更を有効にするために、OpenShift を再起動します。

```
# master-restart api
# master-restart controllers
```

3. デフォルトが追加されていることを確認します。

   a. Pod を作成します。

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

   以下に例を示します。

   ```
   $ oc create -f hello-pod.yaml
   pod "hello-pod" created
   ```

   b. Pod の容認を確認します。

   ```
   $ oc describe pod <pod-name> |grep -i toleration
   ```

   以下に例を示します。

   ```
   $ oc describe pod hello-pod |grep -i toleration
   Tolerations:    node.kubernetes.io/not-ready=:Exists:NoExecute for 300s
   ```

16.11.5. ノード問題の Pod エビクション

OpenShift Container Platform は、**node unreachable** および **node not ready** 状態をテイントとして表示するよう設定できます。これにより、デフォルトの 5 分を使用するのではなく、unreachable (到達不能) または not ready (準備ができていない) 状態になるノードにバインドされたままになる期間を Pod 仕様ごとに指定することができます。

テイントベースのエビクション機能が有効にされた状態で、テイントはノードコントローラーによって自動的に追加され、Pod を **Ready** ノードからエビクトするための通常のロジックは無効にされます。

- ノードが not ready (準備ができていない) 状態になると、**node.kubernetes.io/not-ready:NoExecute** テイントは追加され、Pod はノードでスケジュールできなくなります。既存 Pod は容認期間 (秒数) 中はそのまま残ります。
- ノードが not reachable (到達不能) の状態になると、**node.kubernetes.io/unreachable:NoExecute** テイントは追加され、Pod はノードでスケジュールできません。既存 Pod は容認期間 (秒数) 中はそのまま残ります。

テイントベースのエビクションを有効にするには、以下を実行します。

1. マスター設定ファイル (**/etc/origin/master/master-config.yaml**) を変更して、以下を **kubernetesMasterConfig** セクションに追加します。

```
kubernetesMasterConfig:
  controllerArguments:
```

```
    feature-gates:
    - TaintBasedEvictions=true
```

2. テイントがノードに追加されていることを確認します。

```
$ oc describe node $node | grep -i taint

Taints: node.kubernetes.io/not-ready:NoExecute
```

3. 変更を有効にするために、OpenShift を再起動します。

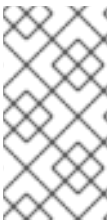
```
# master-restart api
# master-restart controllers
```

4. 容認を Pod に追加します。

```
tolerations:
- key: "node.kubernetes.io/unreachable"
  operator: "Exists"
  effect: "NoExecute"
  tolerationSeconds: 6000
```

または

```
tolerations:
- key: "node.kubernetes.io/not-ready"
  operator: "Exists"
  effect: "NoExecute"
  tolerationSeconds: 6000
```



注記

ノードの問題の発生時に Pod エビクションの既存の レート制限 の動作を維持するために、システムはテイントをレートが制限された方法で追加します。これにより、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 エビクションを防ぐことができます。

16.11.6. Daemonset および容認

Daemonset Pod は、Default Toleration Seconds (デフォルトの容認期間の秒数) 機能が無効にされている場合でも、**tolerationSeconds** のない **node.kubernetes.io/unreachable** および **node.kubernetes.io/not-ready** の **NoExecute** 容認で作成され、DeamonSet Pod がこの問題が原因でエビクトされないようにします。

16.11.7. 例

テイントおよび容認は、Pod をノードから切り離し、ノードで実行されるべきでない Pod をエビクトする柔軟性のある方法として使用できます。以下は典型的なシナリオのいくつかになります。

- ノードをユーザー専用にする
- ユーザーをノードにバインドする

- 特殊ハードウェアを持つノードを専用ノードにする

16.11.7.1. ノードをユーザー専用にする

ノードのセットを特定のユーザーセットが排他的に使用するように指定できます。

専用ノードを指定するには、以下を実行します。

1. 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
```

2. カスタム受付コントローラーを作成して対応する容認を Pod に追加します。
   容認のある Pod のみが専用ノードを使用することを許可されます。

16.11.7.2. ユーザーのノードへのバインド

特定ユーザーが専用ノードのみを使用できるようにノードを設定することができます。

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. 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
```

2. カスタム受付コントローラーを作成して対応する容認を Pod に追加します。
   受付コントローラーは、Pod が **key:value** ラベル (**dedicated=groupName**) が付けられたノードのみにスケジュールされるようにノードのアフィニティーを追加します。

3. テイントと同様のラベル (**key:value** ラベルなど) を専用ノードに追加します。

16.11.7.3. 特殊ハードウェアを持つノード

ノードの小規模なサブセットが特殊ハードウェア (GPU など) を持つクラスターでは、テイントおよび容認を使用して、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。

Pod が特殊ハードウェアからブロックされるようにするには、以下を実行します。

1. 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
$ oc adm taint nodes <node-name> disktype=ssd:PreferNoSchedule
```

2. 受付コントローラー を使用して特殊ハードウェアを使用する Pod に対応する容認を追加します。

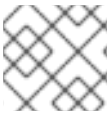
たとえば受付コントローラーは容認を追加することで、Pod の一部の特徴を使用し、Pod が特殊ノードを使用できるかどうかを判別できます。

Pod が特殊ハードウェアのみを使用できるようにするには、追加のメカニズムが必要です。たとえば、特殊ハードウェアを持つノードにラベルを付け、ハードウェアを必要とする Pod でノードのアフィニティーを使用できます。

第17章 クォータの設定

17.1. 概要

ResourceQuota オブジェクトで定義されるリソースクォータは、プロジェクトごとにリソース消費量の総計を制限する制約を指定します。これは、タイプ別にプロジェクトで作成できるオブジェクトの数量を制限すると共に、そのプロジェクトのリソースが消費できるコンピュートリソースおよびストレージの合計量を制限することができます。

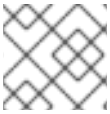


注記

コンピュートリソースについての詳細は、Developer Guide を参照してください。

17.2. クォータで管理されるリソース

以下では、クォータで管理できる一連のコンピュートリソースとオブジェクトタイプについて説明します。



注記

status.phase in (Failed, Succeeded) が true の場合、Pod は終了状態にあります。

表17.1 クォータで管理されるコンピュートリソース

リソース名	説明
cpu	非終了状態のすべての Pod での CPU 要求の合計はこの値を超えることができません。**cpu** および **requests.cpu** は同じ値であり、相互に置き換え可能なものとして使用できます。
memory	非終了状態のすべての Pod でのメモリー要求の合計はこの値を超えることができません。**memory** および **requests.memory** は同じ値であり、相互に置き換え可能なものとして使用できます。
ephemeral-storage	非終了状態のすべての Pod におけるローカルの一時ストレージ要求の合計は、この値を超えることができません。**ephemeral-storage** および **requests.ephemeral-storage** は同じ値であり、相互に置き換え可能なものとして使用できます。このリソースは、一時ストレージのテクノロジープレビュー機能が有効にされている場合にのみ利用できます。この機能はデフォルトでは無効になっています。
requests.cpu	非終了状態のすべての Pod での CPU 要求の合計はこの値を超えることができません。**cpu** および **requests.cpu** は同じ値であり、相互に置き換え可能なものとして使用できます。
requests.memory	非終了状態のすべての Pod でのメモリー要求の合計はこの値を超えることができません。**memory** および **requests.memory** は同じ値であり、相互に置き換え可能なものとして使用できます。

リソース名	説明
requests.ephemeral-storage	非終了状態のすべての Pod における一時ストレージ要求の合計は、この値を超えることができません。**ephemeral-storage** および **requests.ephemeral-storage** は同じ値であり、相互に置き換え可能なものとして使用できます。このリソースは、一時ストレージのテクノロジープレビュー機能が有効にされている場合にのみ利用できます。この機能はデフォルトでは無効になっています。
limits.cpu	非終了状態のすべての Pod での CPU 制限の合計はこの値を超えることができません。
limits.memory	非終了状態のすべての Pod でのメモリー制限の合計はこの値を超えることができません。
limits.ephemeral-storage	非終了状態のすべての Pod における一時ストレージ制限の合計は、この値を超えることができません。このリソースは、一時ストレージのテクノロジープレビュー機能が有効にされている場合にのみ利用できます。この機能はデフォルトでは無効になっています。

表17.2 クォータで管理されるストレージリソース

リソース名	説明
requests.storage	任意の状態のすべての永続ボリューム要求 (PVC) でのストレージ要求の合計は、この値を超えることができません。
persistentvolumeclaims	プロジェクトに存在できる永続ボリューム要求 (PVC) の合計数です。
<storage-class-name>.storageclass.storage.k8s.io/requests.storage	一致するストレージクラスを持つ、任意の状態のすべての永続ボリューム要求 (PVC) でのストレージ要求の合計はこの値を超えることができません。
<storage-class-name>.storageclass.storage.k8s.io/persistentvolumeclaims	プロジェクトに存在できる、一致するストレージクラスを持つ Persistent Volume Claim (永続ボリューム要求、PVC) の合計数です。

表17.3 クォータで管理されるオブジェクト数

リソース名	説明
pods	プロジェクトに存在できる非終了状態の Pod の合計数です。
replicationcontrollers	プロジェクトに存在できるレプリケーションコントローラーの合計数です。

リソース名	説明
resourcequotas	プロジェクトに存在できるリソースクォータの合計数です。
services	プロジェクトに存在できるサービスの合計数です。
secrets	プロジェクトに存在できるシークレットの合計数です。
configmaps	プロジェクトに存在できる **ConfigMap** オブジェクトの合計数です。
persistentvolumeclaims	プロジェクトに存在できる永続ボリューム要求 (PVC) の合計数です。
openshift.io/imagestreams	プロジェクトに存在できるイメージストリームの合計数です。

クォータの作成 時に、**count/<resource>.<group>** 構文を使用して、これらの標準的な namespace を使用しているリソースタイプのオブジェクトカウントクォータを設定できます。

```
$ oc create quota <name> --hard=count/<resource>.<group>=<quota> 1
```

1 **<resource>** はリソースの名前であり、**<group>** は API グループです (該当する場合)。リソースおよびそれらの関連付けられた API グループの一覧に **kubectl api-resources** コマンドを使用します。

17.2.1. 拡張リソースのリソースクォータの設定

リソースのオーバーコミットは拡張リソースには許可されません。そのため、クォータで同じ拡張リソースの **requests** および **limits** を指定する必要があります。現時点で、接頭辞 **requests.** のあるクォータ項目のみが拡張リソースに許可されます。以下は、GPU リソース **nvidia.com/gpu** のリソースクォータを設定する方法についてのシナリオ例です。

手順

1. クラスター内のノードで利用可能な GPU の数を判別します。以下に例を示します。

```
# oc describe node ip-172-31-27-209.us-west-2.compute.internal | egrep
'Capacity|Allocatable|gpu'
                    openshift.com/gpu-accelerator=true
Capacity:
 nvidia.com/gpu:  2
Allocatable:
 nvidia.com/gpu:  2
  nvidia.com/gpu  0          0
```

この例では、2 つの GPU が利用可能です。

2. namespace **nvidia** にクォータを設定します。この例では、クォータは **1** です。

```
# cat gpu-quota.yaml
```

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: gpu-quota
  namespace: nvidia
spec:
  hard:
    requests.nvidia.com/gpu: 1
```

3. クォータを作成します。

```
# oc create -f gpu-quota.yaml
resourcequota/gpu-quota created
```

4. namespace に正しいクォータが設定されていることを確認します。

```
# oc describe quota gpu-quota -n nvidia
Name:                   gpu-quota
Namespace:              nvidia
Resource                Used  Hard
--------                ----  ----
requests.nvidia.com/gpu  0     1
```

5. 単一 GPU を要求する Pod を実行します。

```
# oc create pod gpu-pod.yaml
```

```
apiVersion: v1
kind: Pod
metadata:
  generateName: gpu-pod-
  namespace: nvidia
spec:
  restartPolicy: OnFailure
  containers:
  - name: rhel7-gpu-pod
    image: rhel7
    env:
      - name: NVIDIA_VISIBLE_DEVICES
        value: all
      - name: NVIDIA_DRIVER_CAPABILITIES
        value: "compute,utility"
      - name: NVIDIA_REQUIRE_CUDA
        value: "cuda>=5.0"

    command: ["sleep"]
    args: ["infinity"]

    resources:
      limits:
        nvidia.com/gpu: 1
```

6. Pod が実行されていることを確認します。

```
# oc get pods
NAME            READY    STATUS     RESTARTS   AGE
gpu-pod-s46h7   1/1      Running    0          1m
```

7. クォータ **Used** のカウンターが正しいことを確認します。

```
# oc describe quota gpu-quota -n nvidia
Name:                   gpu-quota
Namespace:              nvidia
Resource                Used  Hard
--------                ----  ----
requests.nvidia.com/gpu  1     1
```

8. **nvidia** namespace で 2 番目の GPU Pod の作成を試行します。2 つの GPU があるので、これをノード上で実行することは可能です。

```
# oc create -f gpu-pod.yaml
Error from server (Forbidden): error when creating "gpu-pod.yaml": pods "gpu-pod-f7z2w" is
forbidden: exceeded quota: gpu-quota, requested: requests.nvidia.com/gpu=1, used:
requests.nvidia.com/gpu=1, limited: requests.nvidia.com/gpu=1
```

クォータが 1 GPU であり、この Pod がそのクォータを超える 2 つ目の GPU の割り当てを試行したため、**Forbidden** エラーメッセージが表示されることが予想されます。

17.3. クォータのスコープ

各クォータには スコープ のセットが関連付けられます。クォータは、列挙されたスコープの交差部分に一致する場合にのみリソースの使用状況を測定します。

スコープをクォータに追加すると、クォータが適用されるリソースのセットを制限できます。許可されるセット以外のリソースを設定すると、検証エラーが発生します。

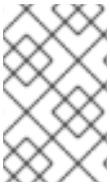
スコープ	説明
Terminating	**spec.activeDeadlineSeconds >= 0** の Pod に一致します。
NotTerminating	**spec.activeDeadlineSeconds** が **nil** の Pod に一致します。
BestEffort	**cpu** または **memory** のいずれかの QoS (Quality of Service) が Best Effort の Pod に一致します。コンピュートリソースのコミットについての詳細は、QoS (Quality of Service) クラス を参照してください。
NotBestEffort	**cpu** および **memory** の QoS (Quality of Service) が Best Effort でない Pod に一致します。

BestEffort スコープは、以下のリソースを制限するようにクォータを制限します。

- **pods**

Terminating、**NotTerminating**、および **NotBestEffort** スコープは、以下のリソースを追跡するようにクォータを制限します。

- **pods**
- **memory**
- **requests.memory**
- **limits.memory**
- **cpu**
- **requests.cpu**
- **limits.cpu**
- **ephemeral-storage**
- **requests.ephemeral-storage**
- **limits.ephemeral-storage**



注記

一時ストレージ要求と制限は、テクノロジープレビューとして提供されている一時ストレージを有効にした場合にのみ適用されます。この機能はデフォルトでは無効になっています。

17.4. クォータの実施

プロジェクトのリソースクォータが最初に作成されると、プロジェクトは、更新された使用状況の統計が計算されるまでクォータ制約の違反を引き起こす可能性のある新規リソースの作成機能を制限します。

クォータが作成され、使用状況の統計が更新されると、プロジェクトは新規コンテンツの作成を許可します。リソースを作成または変更する場合、クォータの使用量はリソースの作成または変更要求があるとすぐに増分します。

リソースを削除する場合、クォータの使用量は、プロジェクトのクォータ統計の次回の完全な再計算時に減分されます。設定可能な時間を指定して、クォータ使用量の統計値を現在確認されるシステム値まで下げるのに必要な時間を決定します。

プロジェクト変更がクォータ使用制限を超える場合、サーバーはそのアクションを拒否し、クォータ制約を違反していること、およびシステムで現在確認される使用量の統計値を示す適切なエラーメッセージがユーザーに返されます。

17.5. 要求 VS 制限

コンピュートリソース の割り当て時に、各コンテナーは CPU、メモリー、一時ストレージそれぞれに要求値と制限値を指定できます。クォータはこれらの値のいずれも制限できます。

クォータに **requests.cpu** または **requests.memory** の値が指定されている場合、すべての着信コンテナーがそれらのリソースを明示的に要求することが求められます。クォータに **limits.cpu** または **limits.memory** の値が指定されている場合、すべての着信コンテナーがそれらのリソースの明示的な制限を指定することが求められます。

17.6. リソースクォータ定義のサンプル

core-object-counts.yaml

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: core-object-counts
spec:
  hard:
    configmaps: "10" (1)
    persistentvolumeclaims: "4" (2)
    replicationcontrollers: "20" (3)
    secrets: "10" (4)
    services: "10" (5)
```

(1) プロジェクトに存在できる **ConfigMap** オブジェクトの合計数です。

(2) プロジェクトに存在できる Persistent Volume Claim (永続ボリューム要求、PVC) の合計数です。

(3) プロジェクトに存在できるレプリケーションコントローラーの合計数です。

(4) プロジェクトに存在できるシークレットの合計数です。

(5) プロジェクトに存在できるサービスの合計数です。

openshift-object-counts.yaml

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: openshift-object-counts
spec:
  hard:
    openshift.io/imagestreams: "10" (1)
```

(1) プロジェクトに存在できるイメージストリームの合計数です。

compute-resources.yaml

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: compute-resources
spec:
  hard:
    pods: "4" (1)
    requests.cpu: "1" (2)
    requests.memory: 1Gi (3)
    requests.ephemeral-storage: 2Gi (4)
    limits.cpu: "2" (5)
    limits.memory: 2Gi (6)
    limits.ephemeral-storage: 4Gi (7)
```

1. プロジェクトに存在できる非終了状態の Pod の合計数です。
2. 非終了状態のすべての Pod において、CPU 要求の合計は 1 コアを超えることができません。
3. 非終了状態のすべての Pod において、メモリー要求の合計は 1 Gi を超えることができません。
4. 非終了状態のすべての Pod において、一時ストレージ要求の合計は 2 Gi を超えることができません。
5. 非終了状態のすべての Pod において、CPU 制限の合計は 2 コアを超えることができません。
6. 非終了状態のすべての Pod において、メモリー制限の合計は 2 Gi を超えることができません。
7. 非終了状態のすべての Pod において、一時ストレージ制限の合計は 4 Gi を超えることができません。

besteffort.yaml

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: besteffort
spec:
  hard:
    pods: "1" (1)
  scopes:
  - BestEffort (2)
```

1. プロジェクトに存在できる QoS (Quality of Service) が **BestEffort** の非終了状態の Pod の合計数です
2. クォータを、メモリーまたは CPU のいずれかの QoS (Quality of Service) が **BestEffort** の一致する Pod のみに制限します。

compute-resources-long-running.yaml

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: compute-resources-long-running
spec:
  hard:
    pods: "4" (1)
    limits.cpu: "4" (2)
    limits.memory: "2Gi" (3)
    limits.ephemeral-storage: "4Gi" (4)
  scopes:
  - NotTerminating (5)
```

1. 非終了状態の Pod の合計数です。
2. 非終了状態のすべての Pod において、CPU 制限の合計はこの値を超えることができません。

3 非終了状態のすべての Pod において、メモリー制限の合計はこの値を超えることができません。

4 非終了状態のすべての Pod において、一時ストレージ制限の合計はこの値を超えることができません。

5 クォータを **spec.activeDeadlineSeconds** が **nil** に設定されている一致する Pod のみに制限します。ビルド Pod は、**RestartNever** ポリシーが適用されない場合に **NotTerminating** になります。

compute-resources-time-bound.yaml

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: compute-resources-time-bound
spec:
  hard:
    pods: "2" (1)
    limits.cpu: "1" (2)
    limits.memory: "1Gi" (3)
    limits.ephemeral-storage: "1Gi" (4)
  scopes:
  - Terminating (5)
```

1 非終了状態の Pod の合計数です。

2 非終了状態のすべての Pod において、CPU 制限の合計はこの値を超えることができません。

3 非終了状態のすべての Pod において、メモリー制限の合計はこの値を超えることができません。

4 非終了状態のすべての Pod において、一時ストレージ制限の合計はこの値を超えることができません。

5 クォータを **spec.activeDeadlineSeconds >=0** に設定されている一致する Pod のみに制限します。たとえば、このクォータはビルド Pod またはデプロイヤー Pod に影響を与えますが、web サーバーまたはデータベースなどの長時間実行されない Pod には影響を与えません。

storage-consumption.yaml

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: storage-consumption
spec:
  hard:
    persistentvolumeclaims: "10" (1)
    requests.storage: "50Gi" (2)
    gold.storageclass.storage.k8s.io/requests.storage: "10Gi" (3)
    silver.storageclass.storage.k8s.io/requests.storage: "20Gi" (4)
    silver.storageclass.storage.k8s.io/persistentvolumeclaims: "5" (5)
    bronze.storageclass.storage.k8s.io/requests.storage: "0" (6)
    bronze.storageclass.storage.k8s.io/persistentvolumeclaims: "0" (7)
```

1. プロジェクト内の永続ボリューム要求 (PVC) の合計数です。
2. 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、要求されるストレージの合計はこの値を超えることができません。
3. 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、gold ストレージクラスで要求されるストレージの合計はこの値を超えることができません。
4. 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、silver ストレージクラスで要求されるストレージの合計はこの値を超えることができません。
5. 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、silver ストレージクラスの要求の合計数はこの値を超えることができません。
6. 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、bronze ストレージクラスで要求されるストレージの合計はこの値を超えることができません。これが **0** に設定される場合、bronze ストレージクラスはストレージを要求できないことを意味します。
7. 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、bronze ストレージクラスで要求されるストレージの合計はこの値を超えることができません。これが **0** に設定される場合は、bronze ストレージクラスでは要求を作成できないことを意味します。

17.7. クォータの作成

クォータを作成するには、リソースクォータ定義のサンプル に示されるように、まずクォータをファイルに定義します。次に、そのファイルを使用してこれをプロジェクトに適用します。

```
$ oc create -f <resource_quota_definition> [-n <project_name>]
```

以下に例を示します。

```
$ oc create -f core-object-counts.yaml -n demoproject
```

17.7.1. オブジェクトカウントクォータの作成

BuildConfig および **DeploymentConfig** などの、OpenShift Container Platform の標準的な namespace を使用しているリソースタイプのすべてに オブジェクトカウントクォータ を作成できます。オブジェクトクォータカウントは、定義されたクォータをすべての標準的な namespace を使用しているリソースタイプに設定します。

リソースクォータの使用時に、オブジェクトがサーバーストレージにある場合、そのオブジェクトはクォータに基づいてチャージされます。以下のクォータのタイプはストレージリソースが使い切られることから保護するのに役立ちます。

リソースのオブジェクトカウントクォータを設定するには、以下のコマンドを実行します。

```
$ oc create quota <name> --hard=count/<resource>.<group>=<quota>,count/<resource>.<group>=
<quota>
```

以下に例を示します。

```
$ oc create quota test --
hard=count/deployments.extensions=2,count/replicasets.extensions=4,count/pods=3,count/secrets=4
```

```
resourcequota "test" created

$ oc describe quota test
Name:                       test
Namespace:                  quota
Resource                    Used  Hard
--------                    ----  ----
count/deployments.extensions  0     2
count/pods                    0     3
count/replicasets.extensions  0     4
count/secrets                 0     4
```

この例では、一覧表示されたリソースをクラスター内の各プロジェクトのハード制限に制限します。

17.8. クォータの表示

web コンソールでプロジェクトの **Quota** ページに移動し、プロジェクトのクォータで定義されるハード制限に関連する使用状況の統計を表示できます。

CLI を使用してクォータの詳細を表示することもできます。

1. 最初に、プロジェクトで定義されたクォータの一覧を取得します。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 quota -n demoproject
NAME                AGE
besteffort          11m
compute-resources   2m
core-object-counts  29m
```

2. 次に、関心のあるクォータについて記述します。たとえば、**core-object-counts** クォータの場合、以下を実行します。

```
$ oc describe quota core-object-counts -n demoproject
Name:   core-object-counts
Namespace:  demoproject
Resource  Used Hard
--------  ---- ----
configmaps  3 10
persistentvolumeclaims 0 4
replicationcontrollers 3 20
secrets   9 10
services  2 10
```

17.9. クォータの同期期間の設定

リソースのセットが削除される際に、リソースの同期期間が **/etc/origin/master/master-config.yaml** ファイルの **resource-quota-sync-period** 設定によって決定されます。

クォータの使用状況が復元される前に、ユーザーがリソースの再使用を試行すると問題が発生する場合があります。**resource-quota-sync-period** 設定を変更して、リソースセットの再生成が所定の期間 (秒単位) に実行され、リソースを再度利用可能にすることができます。

```
kubernetesMasterConfig:
```

```
apiLevels:
- v1beta3
- v1
apiServerArguments: null
controllerArguments:
  resource-quota-sync-period:
    - "10s"
```

変更後に、マスターサービスを再起動してそれらの変更を適用します。

```
# master-restart api
# master-restart controllers
```

再生成時間の調整は、リソースの作成および自動化が使用される場合のリソース使用状況の判別に役立ちます。

注記

resource-quota-sync-period 設定は、システムパフォーマンスのバランスを取るように設計されています。同期期間を短縮すると、マスターに大きな負荷がかかる可能性があります。

17.10. デプロイメント設定におけるクォータアカウンティング

クォータがプロジェクトに定義されている場合、デプロイメント設定の考慮事項については、デプロイメントリソース を参照してください。

17.11. リソース消費における明示的なクォータの要求

リソースがクォータで管理されていない場合、ユーザーには消費できるリソース量の制限がありません。たとえば、gold ストレージクラスに関連するストレージのクォータがない場合、プロジェクトが作成できる gold ストレージの容量はバインドされません。

高コストのコンピュートまたはストレージリソースの場合、管理者はリソースを消費するための明示的なクォータの付与が必要となるようにする場合があります。たとえば、プロジェクトに gold ストレージクラスに関連するストレージのクォータが明示的に付与されていない場合、そのプロジェクトのユーザーはこのタイプのストレージを作成することができません。

特定リソースの消費における明示的なクォータが必要となるようにするには、以下のスタンザを master-config.yaml に追加する必要があります。

```
admissionConfig:
  pluginConfig:
    ResourceQuota:
      configuration:
        apiVersion: resourcequota.admission.k8s.io/v1alpha1
        kind: Configuration
        limitedResources:
        - resource: persistentvolumeclaims (1)
        matchContains:
        - gold.storageclass.storage.k8s.io/requests.storage (2)
```

(1) デフォルトで消費が制限されるグループ/リソースです。

2 デフォルトで制限対象となる、グループ/リソースに関連付けられたクォータで追跡されるリソースの名前です。

上記の例では、クォータシステムは **PersistentVolumeClaim** を作成するか、または更新するすべての操作をインターセプトします。これは、クォータで認識されるリソースが消費されることを確認し、プロジェクトのそれらのリソースのクォータがない場合に要求は拒否されます。この例ではユーザーが gold ストレージクラスに関連付けられたストレージを使用する **PersistentVolumeClaim** を作成しており、プロジェクトに一致するクォータがない場合には要求が拒否されます。

17.12. 既知の問題

- 無効なオブジェクトにより、プロジェクトのクォータリソースが使い切られる可能性があります。クォータはリソースの検証前に受付において増分します。その結果、クォータは Pod が最終的に永続しない場合でも増分する可能性があります。この問題は、今後のリリースで解決される予定です。(BZ1485375)

第18章 複数プロジェクトのクォータ設定

18.1. 概要

ClusterResourceQuota オブジェクトで定義される複数プロジェクトのクォータは、複数プロジェクト間で クォータ を共有できるようにします。それぞれの選択されたプロジェクトで使用されるリソースは集計され、その集計は選択したすべてのプロジェクトでリソースを制限するために使用されます。

18.2. プロジェクトの選択

プロジェクトは、アノテーションの選択またはラベルの選択のいずれか、またはその両方に基づいて選択できます。たとえば、アノテーションに基づいてプロジェクトを選択するには、以下のコマンドを実行します。

```
$ oc create clusterquota for-user \
    --project-annotation-selector openshift.io/requester=<user-name> \
    --hard pods=10 \
    --hard secrets=20
```

これは以下の **ClusterResourceQuota** オブジェクトを作成します。

```
apiVersion: v1
kind: ClusterResourceQuota
metadata:
  name: for-user
spec:
  quota: (1)
    hard:
      pods: "10"
      secrets: "20"
  selector:
    annotations: (2)
      openshift.io/requester: <user-name>
    labels: null (3)
status:
  namespaces: (4)
  - namespace: ns-one
    status:
      hard:
        pods: "10"
        secrets: "20"
      used:
        pods: "1"
        secrets: "9"
  total: (5)
    hard:
      pods: "10"
      secrets: "20"
    used:
      pods: "1"
      secrets: "9"
```

(1) 選択したプロジェクトに対して実施される **ResourceQuotaSpec** オブジェクトです。

2 アノテーションの単純なキー/値のセレクターです。

3 プロジェクトを選択するために使用できるラベルセレクターです。

4 選択された各プロジェクトの現在のクォータの使用状況を記述する namespace ごとのマップです。

5 選択されたすべてのプロジェクトにおける使用量の総計です。

この複数プロジェクトのクォータの記述は、デフォルトのプロジェクト要求エンドポイントを使用して **<user-name>** によって要求されるすべてのプロジェクトを制御します。ここでは、10 Pod および 20 シークレットに制限されます。

同様にラベルに基づいてプロジェクトを選択するには、以下のコマンドを実行します。

```
$ oc create clusterresourcequota for-name \ 1
    --project-label-selector=name=frontend \ 2
    --hard=pods=10 --hard=secrets=20
```

1 **clusterresourcequota** および **clusterquota** はいずれも同じコマンドのエイリアスです。**for-name** は **clusterresourcequota** オブジェクトの名前です。

2 ラベル別にプロジェクトを選択するには、**--project-label-selector=key=value** 形式を使用してキーと値のペアを指定します。

これは以下の **ClusterResourceQuota** オブジェクト定義を作成します。

```
apiVersion: v1
kind: ClusterResourceQuota
metadata:
  creationTimestamp: null
  name: for-name
spec:
  quota:
    hard:
      pods: "10"
      secrets: "20"
  selector:
    annotations: null
    labels:
      matchLabels:
        name: frontend
```

18.3. 適用可能な CLUSTERRESOURCEQUOTAS の表示

プロジェクト管理者は、各自のプロジェクトを制限する複数プロジェクトのクォータを作成したり、変更したりすることはできませんが、それぞれのプロジェクトに適用される複数プロジェクトのクォータを表示することはできます。プロジェクト管理者は、**AppliedClusterResourceQuota** リソースを使ってこれを実行できます。

```
$ oc describe AppliedClusterResourceQuota
```

以下が生成されます。

```
Name:   for-user
Namespace:  <none>
Created:  19 hours ago
Labels:   <none>
Annotations:  <none>
Label Selector: <null>
AnnotationSelector: map[openshift.io/requester:<user-name>]
Resource  Used  Hard
--------  ----  ----
pods        1     10
secrets     9     20
```

18.4. 選択における粒度

クォータの割り当てを要求する際にロックに関して考慮する必要があるため、複数プロジェクトのクォータで選択されるアクティブなプロジェクトの数は重要な考慮点になります。単一の複数プロジェクトクォータで 100 を超えるプロジェクトを選択すると、それらのプロジェクトの API サーバーの応答に負の影響が及びます。

第19章 オブジェクトのプルーニング

19.1. 概要

OpenShift Container Platform で作成される API オブジェクト は、一定期間が経過すると、アプリケーションのビルドやデプロイなどの通常のユーザー操作によって etcd データストア に蓄積される可能性があります。

管理者は、不要になった古いバージョンのオブジェクトを OpenShift Container Platform インスタンスから定期的にプルーニングできます。たとえば、イメージのプルーニングにより、使用されなくなったものの、ディスク領域を使用している古いイメージや層を削除できます。

19.2. プルーニングの基本操作

CLI は、共通の親コマンドでプルーニング操作を分類します。

```
$ oc adm prune <object_type> <options>
```

これにより、以下が指定されます。

- **groups**、**builds**、**deployments**、または **images** などのアクションを実行するための **<object_type>**。
- オブジェクトタイプのプルーニングの実行においてサポートされる **<options>**。

19.3. グループのプルーニング

グループのレコードを外部プロバイダーからプルーニングするために、管理者は以下のコマンドを実行できます。

```
$ oc adm prune groups --sync-config=path/to/sync/config [<options>]
```

表19.1 グループのプルーニング用の CLI の設定オプション

オプション	説明
--confirm	ドライランを実行する代わりにプルーニングが実行されることを示します。
--blacklist	グループのブラックリストファイルへのパスです。ブラックリストファイルの構造については、Syncing Groups With LDAPを参照してください。
--whitelist	グループのホワイトリストファイルへのパスです。ホワイトリストファイルの構造については、Syncing Groups With LDAPを参照してください。
--sync-config	同期設定ファイルへのパスです。このファイルの構造については、Configuring LDAP Sync を参照してください。

prune コマンドが削除するグループを表示するには、以下を実行します。

```
$ oc adm prune groups --sync-config=ldap-sync-config.yaml
```

prune 操作を実行するには、以下を実行します。

```
$ oc adm prune groups --sync-config=ldap-sync-config.yaml --confirm
```

19.4. デプロイメントのプルーニング

使用年数やステータスによりシステムで不要となったデプロイメントをプルーニングするために、管理者は以下のコマンドを実行できます。

```
$ oc adm prune deployments [<options>]
```

表19.2 デプロイメントのプルーニング用の CLI の設定オプション

オプション	説明
--confirm	ドライランを実行する代わりにプルーニングが実行されることを示します。
--orphans	デプロイメント設定が存在せず、ステータスが complete (完了) または failed (失敗) で、レプリカ数がゼロであるすべてのデプロイメントをプルーニングします。
--keep-complete=<N>	デプロイメント設定に基づき、ステータスが complete (完了) で、レプリカ数がゼロである最後の N デプロイメントを保持します (デフォルト: **5**)。
--keep-failed=<N>	デプロイメント設定に基づき、ステータスが failed (失敗) で、レプリカ数がゼロである最後の N デプロイメントを保持します (デフォルト: **1**)。
--keep-younger-than=<duration>	現在の時間との対比で **<duration>** より後の新しいオブジェクトはプルーニングしません (デフォルト: **60m**)。有効な測定単位には、ナノ秒 (**ns**)、マイクロ秒 (**us**)、ミリ秒 (**ms**)、秒 (**s**)、分 (**m**)、および時間 (**h**) が含まれます。

プルーニング操作によって削除されるものを確認するには、以下を実行します。

```
$ oc adm prune deployments --orphans --keep-complete=5 --keep-failed=1 \
    --keep-younger-than=60m
```

プルーニング操作を実際に実行するには、以下を実行します。

```
$ oc adm prune deployments --orphans --keep-complete=5 --keep-failed=1 \
    --keep-younger-than=60m --confirm
```

19.5. ビルドのプルーニング

使用年数やステータスによりシステムで不要となったビルドをプルーニングするために、管理者は以下のコマンドを実行できます。

```
$ oc adm prune builds [<options>]
```

表19.3 ビルドのプルーニング用の CLI の設定オプション

オプション	説明
--confirm	ドライランを実行する代わりにプルーニングが実行されることを示します。
--orphans	ビルド設定が存在せず、ステータスが complete (完了)、failed (失敗)、error (エラー)、または canceled (中止) のすべてのビルドをプルーニングします。
--keep-complete=<N>	ビルド設定に基づき、ステータスが complete (完了) の最後の N ビルドを保持します (デフォルト: **5**)。
--keep-failed=<N>	ビルド設定に基づき、ステータスが failed (失敗)、error (エラー)、または canceled (中止) の最後の N ビルドを保持します (デフォルト: **1**)。
--keep-younger-than=<duration>	現在の時間との対比で **<duration>** 未満の新しいオブジェクトはプルーニングしません (デフォルト: **60m**)。

プルーニング操作によって削除されるものを確認するには、以下を実行します。

```
$ oc adm prune builds --orphans --keep-complete=5 --keep-failed=1 \
    --keep-younger-than=60m
```

プルーニング操作を実際に実行するには、以下を実行します。

```
$ oc adm prune builds --orphans --keep-complete=5 --keep-failed=1 \
    --keep-younger-than=60m --confirm
```

注記

開発者は、ビルド設定を変更することにより、自動ビルドプルーニング を有効にできます。

19.6. イメージのプルーニング

使用年数やステータスまたは制限の超過によりシステムで不要となったイメージをプルーニングするために、管理者は以下のコマンドを実行できます。

```
$ oc adm prune images [<options>]
```

注記

現時点でイメージをプルーニングするには、まず アクセストークン を使って、ユーザーとして CLI にログイン する必要があります。ユーザーには クラスターロール **system:image-pruner** 以上のロールがなければなりません (例: **cluster-admin**)。

注記

--prune-registry=false が使用されていない限り、イメージのプルーニングにより、統合レジストリーのデータが削除されます。この操作が適切に機能するには、**storage:delete:enabled** が **true** に設定された状態で レジストリーが設定されていること を確認してください。

注記

--namespace フラグの付いたイメージをプルーニングしてもイメージは削除されず、イメージストリームのみが削除されます。イメージは namespace を使用しないリソースです。そのため、プルーニングを特定の namespace に制限すると、イメージの現在の使用量を算出できなくなります。

デフォルトで、統合レジストリーは Blob メタデータをキャッシュしてストレージに対する要求数を減らし、要求の処理速度を高めます。プルーニングによって統合レジストリーのキャッシュが更新されることはありません。プルーニング後にプッシュされる、プルーニングされた層を含むイメージは破損します。キャッシュにメタデータを持つプルーニングされた層はプッシュされないためです。これは、レジストリーの再デプロイによって実行できます。

```
$ oc rollout latest dc/docker-registry
```

統合レジストリーが redis キャッシュ を使用する場合は、データベースを手動でクリーンアップする必要があります。

プルーニング後にレジストリーを再デプロイすることがオプションでない場合は、キャッシュを永続的に無効にする 必要があります。

表19.4 イメージのプルーニング用の CLI の設定オプション

オプション	説明
--all	レジストリーにプッシュされていないものの、プルスルー (pullthrough) でミラーリングされたイメージを組み込みます。これはデフォルトでオンに設定されます。プルーニングを統合レジストリーにプッシュされたイメージに制限するには、**--all=false** を渡します。
--certificate-authority	OpenShift Container Platform で管理されるレジストリーと通信する際に使用する認証局ファイルへのパスです。デフォルトは現行ユーザーの設定ファイルの認証局データに設定されます。これが指定されている場合、セキュアな通信が実行されます。
--confirm	ドライランを実行する代わりにプルーニングが実行されることを示します。これには、統合コンテナーイメージレジストリーへの有効なルートが必要になります。このコマンドがクラスターネットワーク外で実行される場合、ルートは **--registry-url** を使用して指定される必要があります。

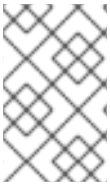
オプション	説明
--force-insecure	**このオプションは注意して使用してください。**HTTP 経由でホストされているか、または無効な HTTPS 証明書を持つ Docker レジストリーへの非セキュアな接続を許可します。詳細は、セキュアまたは非セキュアな接続の使用 を参照してください。
--keep-tag-revisions=<N>	それぞれのイメージストリームについては、タグごとに最大 N のイメージリビジョンを保持します (デフォルト: **3**)。
--keep-younger-than=<duration>	現在の時間との対比で **<duration>** 未満の新しいイメージはプルーニングしません。現在の時間との対比で **<duration>** 未満の他のオブジェクトで参照されるイメージはプルーニングしません (デフォルト: 60m)。
--prune-over-size-limit	同じプロジェクトに定義される最小の 制限 を超える各イメージをプルーニングします。このフラグは **--keep-tag-revisions** または **--keep-younger-than** と共に使用することはできません。
--registry-url	レジストリーと通信する際に使用するアドレスです。このコマンドは、管理されるイメージおよびイメージストリームから判別されるクラスター内の URL の使用を試行します。これに失敗する (レジストリーを解決できないか、これにアクセスできない) 場合、このフラグを使用して他の機能するルートを指定する必要があります。レジストリーのホスト名の前には、特定の接続プロトコルを実施する **https://** または **http://** を付けることができます。
--prune-registry	他のオプションで規定される条件と共に、このオプションは、OpenShift Container Platform イメージ API オブジェクトに対応するレジストリーのデータがプルーニングされるかどうかを制御します。デフォルトで、イメージのプルーニングは、イメージ API オブジェクトとレジストリーの対応するデータの両方を処理します。このオプションは、イメージオブジェクトの数を減らすなどの目的で etcd の内容のみを削除することを検討していて、レジストリーのストレージのクリーンアップは検討していない場合や、レジストリーの適切なメンテナーンス期間中などに レジストリーのハードプルーニング によってこれを別途実行しようとする場合に役立ちます。

19.6.1. イメージのプルーニングの各種条件

- **--keep-younger-than** 分前よりも後に作成され、現時点で以下によって参照されていない OpenShift Container Platform で管理されるイメージ (アノテーション **openshift.io/image.managed** を持つイメージ) を削除します。
  - **--keep-younger-than** 分前よりも後に作成された Pod
  - **--keep-younger-than** 分前よりも後に作成されたイメージストリーム
  - 実行中の Pod
  - 保留中の Pod
  - レプリケーションコントローラー

- デプロイメント設定
- ビルド設定
- ビルド
- **stream.status.tags[].items** の **--keep-tag-revisions** の最新のアイテム

- 同じプロジェクトで定義される最小の制限を超えており、現時点で以下によって参照されていない OpenShift Container Platform で管理されるイメージ (アノテーション**openshift.io/image.managed** を持つイメージ) を削除します。
  - 実行中の Pod
  - 保留中の Pod
  - レプリケーションコントローラー
  - デプロイメント設定
  - ビルド設定
  - ビルド
- 外部レジストリーからのプルーニングはサポートされていません。
- イメージがプルーニングされる際、イメージのすべての参照は **status.tags** にイメージの参照を持つすべてのイメージストリームから削除されます。
- イメージによって参照されなくなったイメージ層も削除されます。



注記

--prune-over-size-limit は **--keep-tag-revisions** または **--keep-younger-than** フラグと共に使用することができません。これを実行すると、この操作が許可されないことを示す情報が返されます。

注記

--prune-registry=false を使用してレジストリーからの OpenShift Container Platform Image API オブジェクトとイメージデータの削除を分離した後に レジストリーのハードプルーニング を実行することで、タイミングウィンドウが部分的に制限され、1 つのコマンドで両方をプルーニングする場合よりも安全に実行できるようになります。ただし、タイミングウィンドウを完全に取り除くことはできません。

たとえばプルーニングの実行時にプルーニング対象のイメージを特定する場合も、そのイメージを参照する Pod を引き続き作成することができます。また、プルーニングの操作時にイメージを参照している可能性のある API オブジェクトを追跡することもできます。 これにより、削除されたコンテンツの参照に関連して発生する可能性のある問題を軽減することができます。

また、**--prune-registry** オプションを指定しないか、または **--prune-registry=true** を指定してプルーニングを再実行しても、**--prune-registry=false** を指定して以前にプルーニングされたイメージの、イメージレジストリー内で関連付けられたストレージがプルーニングされる訳ではないことに注意してください。**--prune-registry=false** を指定してプルーニングされたすべてのイメージは、レジストリーのハードプルーニング によってのみ削除できます。

プルーニング操作によって削除されるものを確認するには、以下を実行します。

1. 最高 3 つのタグリビジョンを保持し、6 分前よりも後に作成されたリソース (イメージ、イメージストリームおよび Pod) を保持します。

```
$ oc adm prune images --keep-tag-revisions=3 --keep-younger-than=60m
```

2. 定義された制限を超えるすべてのイメージをプルーニングします。

```
$ oc adm prune images --prune-over-size-limit
```

前述のオプションでプルーニング操作を実際に実行するには、以下を実行します。

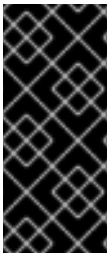
```
$ oc adm prune images --keep-tag-revisions=3 --keep-younger-than=60m --confirm

$ oc adm prune images --prune-over-size-limit --confirm
```

19.6.2. セキュアまたは非セキュアな接続の使用

セキュアな通信の使用は優先され、推奨される方法です。これは、必須の証明書検証と共に HTTPS 経由で実行されます。**prune** コマンドは、可能な場合は常にセキュアな通信の使用を試行します。これを使用できない場合には、非セキュアな通信にフォールバックすることがあり、これには危険が伴います。この場合、証明書検証は省略されるか、または単純な HTTP プロトコルが使用されます。

非セキュアな通信へのフォールバックは、**--certificate-authority** が指定されていない場合、以下のケースで可能になります。

1. **prune** コマンドが **--force-insecure** オプションと共に実行される。
2. 指定される **registry-url** の前に **http://** スキームが付けられる。
3. 指定される **registry-url** がローカルリンクアドレスまたは localhost である。
4. 現行ユーザーの設定が非セキュアな接続を許可する。これは、ユーザーが **--insecure-skip-tls-verify** を使用してログインするか、またはプロンプトが出される際に非セキュアな接続を選択することによって生じる可能性があります。



重要

レジストリーのセキュリティーが、OpenShift Container Platform で使用されるものとは異なる認証局で保護される場合、これを **--certificate-authority** フラグを使用して指定する必要があります。そうしないと、**prune** コマンドは、正しくない認証局の使用 または セキュリティーが保護されたレジストリーに対する非セキュアな接続の使用 で一覧表示されているエラーと同様のエラーを出して失敗します。

19.6.3. イメージのプルーニングに関する問題

イメージがプルーニングされない
イメージが蓄積し続け、**prune** コマンドが予想よりも小規模な削除を実行する場合、プルーニングの候補と見なされるためにイメージに適用する必要のある 条件 を理解していることを確認してください。

とくに削除する必要のあるイメージが、それぞれの タグ履歴 において選択したタグリビジョンのしきい値よりも高い位置にあることを確認します。たとえば、**sha:abz** という名前の古く陳腐化したイメージがあるとします。イメージがタグ付けされている namespace **N** で以下のコマンドを実行すると、イ

メージが **myapp** という単一イメージストリームで 3 回タグ付けされていることがわかります。

```
$ image_name="sha:abz"
$ oc get is -n openshift -o go-template='{{range $isi, $is := .items}}{{range $ti, $tag := $is.status.tags}}
{{range $ii, $item := $tag.items}}{{if eq $item.image "'$image_name'"}}{{$is.metadata.name}}:
{{$tag.tag}} at position {{$ii}} out of {{len $tag.items}}
{{end}}{{end}}{{end}}{{end}}' # Before this place {{end}}{{end}}{{end}}{{end}}, use new line
myapp:v2 at position 4 out of 5
myapp:v2.1 at position 2 out of 2
myapp:v2.1-may-2016 at position 0 out of 1
```

デフォルトオプションが使用される場合、イメージは **myapp:v2.1-may-2016** タグの履歴の **0** の位置にあるためプルーニングされません。イメージがプルーニングの対象と見なされるようにするには、管理者は以下を実行する必要があります。

1. **oc adm prune images** コマンドで **--keep-tag-revisions=0** を指定します。

   注意

   このアクションを実行すると、イメージが指定されたしきい値よりも新しいか、またはこれよりも新しいオブジェクトによって参照されていない限り、すべてのタグが基礎となるイメージと共にすべての namespace から削除されます。

2. リビジョンのしきい値の下にあるすべての istags、つまり **myapp:v2.1** および **myapp:v2.1-may-2016** を削除します。

3. 同じ istag にプッシュする新規ビルドを実行するか、または他のイメージをタグ付けしてイメージを履歴内でさらに移動させます。ただし、これは古いリリースタグの場合には常に適切な操作となる訳ではありません。

特定のイメージのビルド日時が名前の一部になっているタグは、その使用を避ける必要があります (イメージが未定義の期間保持される必要がある場合を除きます)。このようなタグは履歴内で 1 つのイメージのみに関連付けられる可能性があり、その場合にこれらをプルーニングできなくなります。istag 命名の詳細を参照してください。

非セキュアなレジストリーに対するセキュアな接続の使用
oc adm prune images の出力で以下のようなメッセージが表示される場合、レジストリーのセキュリティーは保護されておらず、**oc adm prune images** クライアントがセキュアな接続の使用を試行することを示しています。

```
error: error communicating with registry: Get https://172.30.30.30:5000/healthz: http: server gave
HTTP response to HTTPS client
```

1. 推奨される解決法は、レジストリーのセキュリティーを保護 することです。これが必要でない場合には、**--force-insecure** をコマンドに追加して、クライアントに対して非セキュアな接続の使用を強制することができます (これは推奨される方法ではありません)。

19.6.3.1. セキュリティーが保護されたレジストリーに対する非セキュアな接続の使用

oc adm prune images コマンドの出力に以下のエラーのいずれかが表示される場合、レジストリーのセキュリティー保護に使用されている認証局で署名された証明書が、接続の検証用に **oc adm prune images** クライアントで使用されるものとは異なることを意味します。

```
error: error communicating with registry: Get http://172.30.30.30:5000/healthz: malformed HTTP
```

```
response "\x15\x03\x01\x00\x02\x02"
error: error communicating with registry: [Get https://172.30.30.30:5000/healthz: x509: certificate
signed by unknown authority, Get http://172.30.30.30:5000/healthz: malformed HTTP response
"\x15\x03\x01\x00\x02\x02"]
```

デフォルトでは、ユーザーの接続ファイルに保存されている認証局データが使用されます。これはマスター API との通信の場合も同様です。

--certificate-authority オプションを使用してコンテナーイメージレジストリーサーバーに適切な認証局を指定します。

正しくない認証局の使用
以下のエラーは、セキュリティーが保護されたコンテナーイメージレジストリーの証明書の署名に使用される認証局がクライアントで使用される認証局とは異なることを示しています。

```
error: error communicating with registry: Get https://172.30.30.30:5000/: x509: certificate signed by
unknown authority
```

フラグ **--certificate-authority** を使用して適切な認証局を指定します。

回避策として、**--force-insecure** フラグを代わりに追加することもできます **(**推奨される方法ではありません**)**。

19.7. レジストリーのハードプルーニング

OpenShift Container レジストリーは、OpenShift Container Platform クラスターの etcd で参照されない Blob を蓄積します。基本的なイメージプルーニングの手順はこれらに対応しません。これらの Blob は 孤立した **Blob** と呼ばれています。

孤立した Blob は以下のシナリオで発生する可能性があります。

- **oc delete image <sha256:image-id>** コマンドを使ってイメージを手動で削除すると、etcd のイメージのみが削除され、レジストリーのストレージからは削除されません。
- **docker** デーモンの障害によって生じるレジストリーへのプッシュにより、一部の Blob はアップロードされるものの、(最後のコンポーネントとしてアップロードされる) イメージマニフェスト はアップロードされない。固有のイメージ Blob すべてが孤立します。
- OpenShift Container Platform がクォータの制限によりイメージを拒否します。
- 標準のイメージプルーナーがイメージマニフェストを削除するが、関連する Blob を削除する前に中断されます。
- 対象の Blob を削除できないというレジストリープルーナーのバグにより、それらを参照するイメージオブジェクトは削除されるが、Blob は孤立する。

基本的なイメージプルーニングとは異なるレジストリーの ハードプルーニング により、孤立した Blob を削除することができます。OpenShift Container レジストリーのストレージ領域が不足している場合や、孤立した Blob があると思われる場合にはハードプルーニングを実行する必要があります。

これは何度も行う操作ではなく、多数の孤立した Blob が新たに作成されているという証拠がある場合にのみ実行する必要があります。または、(作成されるイメージの数によって異なりますが) 1 日 1 回などの定期的な間隔で標準のイメージプルーニングを実行することもできます。

孤立した Blob をレジストリーからハードプルーニングするには、以下を実行します。

1. ログイン: CLI を使用し、アクセストークン を持つユーザーとしてログインします。

2. 基本的なイメージプルーニングの実行: 基本的なイメージプルーニングにより、不要になった追加のイメージが削除されます。ハードプルーニングによってイメージが削除される訳ではなく、レジストリーストレージに保存された Blob のみが削除されます。レジストリーストレージに保存された Blob のみが削除されます。したがって、ハードプルーニングの実行前にこれを実行する必要があります。
手順については、イメージのプルーニングを参照してください。

3. レジストリーの読み取り専用モードへの切り替え: レジストリーが読み取り専用モードで実行されていない場合、プルーニングと同時に実行されているプッシュの結果は以下のいずれかになります。

   - 失敗する。 さらに孤立した Blob が新たに発生する。
   - 成功する。 ただし、(参照される Blob の一部が削除されたため) イメージをプルできない。

   プッシュは、レジストリーが読み取り書き込みモードに戻されるまで成功しません。したがって、ハードプルーニングは注意してスケジューリングする必要があります。

   レジストリーを読み取り専用モードに切り換えるには、以下を実行します。

   a. 以下の環境変数を設定します。

   ```
   $ oc set env -n default \
       dc/docker-registry \
       'REGISTRY_STORAGE_MAINTENANCE_READONLY={"enabled":true}'
   ```

   b. デフォルトで、レジストリーは直前の手順が完了すると自動的に再デプロイするはずです。ただし、これらのトリガーを無効にしている場合は、レジストリーを手動で再デプロイし、新規の環境変数が選択されるようにする必要があります。

   ```
   $ oc rollout -n default \
       latest dc/docker-registry
   ```

4. **system:image-pruner** ロールの追加: 一部のリソースを一覧表示するには、レジストリーインスタンスの実行に使用するサービスアカウントに追加のパーミッションが必要になります。

   a. サービスアカウント名を取得します。

   ```
   $ service_account=$(oc get -n default \
       -o jsonpath=$'system:serviceaccount:{.metadata.namespace}:
   {.spec.template.spec.serviceAccountName}\n' \
       dc/docker-registry)
   ```

   b. **system:image-pruner** クラスター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 \
       system:image-pruner \
       ${service_account}
   ```

5. (オプション) プルーナーのドライランモードでの実行: 削除される Blob の数を確認するには、ドライランモードでハードプルーナーを実行します。実際の変更は加えられません。

   ```
   $ oc -n default \
   ```

```
    exec -i -t "$(oc -n default get pods -l deploymentconfig=docker-registry \
    -o jsonpath=$'{.items[0].metadata.name}\n')" \
    -- /usr/bin/dockerregistry -prune=check
```

または、プルーニング候補の実際のパスを取得するには、ロギングレベルを上げます。

```
$ oc -n default \
    exec "$(oc -n default get pods -l deploymentconfig=docker-registry \
      -o jsonpath=$'{.items[0].metadata.name}\n')" \
    -- /bin/sh \
    -c 'REGISTRY_LOG_LEVEL=info /usr/bin/dockerregistry -prune=check'
```

出力サンプル (切り捨て済み)

```
$ oc exec docker-registry-3-vhndw \
    -- /bin/sh -c 'REGISTRY_LOG_LEVEL=info /usr/bin/dockerregistry -prune=check'

time="2017-06-22T11:50:25.066156047Z" level=info msg="start prune (dry-run mode)"
distribution_version="v2.4.1+unknown" kubernetes_version=v1.6.1+$Format:%h$
openshift_version=unknown
time="2017-06-22T11:50:25.092257421Z" level=info msg="Would delete blob:
sha256:00043a2a5e384f6b59ab17e2c3d3a3d0a7de01b2cabeb606243e468acc663fa5"
go.version=go1.7.5 instance.id=b097121c-a864-4e0c-ad6c-cc25f8fdf5a6
time="2017-06-22T11:50:25.092395621Z" level=info msg="Would delete blob:
sha256:0022d49612807cb348cabc562c072ef34d756adfe0100a61952cbcb87ee6578a"
go.version=go1.7.5 instance.id=b097121c-a864-4e0c-ad6c-cc25f8fdf5a6
time="2017-06-22T11:50:25.092492183Z" level=info msg="Would delete blob:
sha256:0029dd4228961086707e53b881e25eba0564fa80033fbbb2e27847a28d16a37c"
go.version=go1.7.5 instance.id=b097121c-a864-4e0c-ad6c-cc25f8fdf5a6
time="2017-06-22T11:50:26.673946639Z" level=info msg="Would delete blob:
sha256:ff7664dfc213d6cc60fd5c5f5bb00a7bf4a687e18e1df12d349a1d07b2cf7663"
go.version=go1.7.5 instance.id=b097121c-a864-4e0c-ad6c-cc25f8fdf5a6
time="2017-06-22T11:50:26.674024531Z" level=info msg="Would delete blob:
sha256:ff7a933178ccd931f4b5f40f9f19a65be5eeeec207e4fad2a5bafd28afbef57e"
go.version=go1.7.5 instance.id=b097121c-a864-4e0c-ad6c-cc25f8fdf5a6
time="2017-06-22T11:50:26.674675469Z" level=info msg="Would delete blob:
sha256:ff9b8956794b426cc80bb49a604a0b24a1553aae96b930c6919a6675db3d5e06"
go.version=go1.7.5 instance.id=b097121c-a864-4e0c-ad6c-cc25f8fdf5a6
...
Would delete 13374 blobs
Would free up 2.835 GiB of disk space
Use -prune=delete to actually delete the data
```

6. ハードプルーニングの実行: ハードプルーニングを実行するには、**docker-registry** Pod の実行中インスタンスで以下のコマンドを実行します。

```
$ oc -n default \
    exec -i -t "$(oc -n default get pods -l deploymentconfig=docker-registry -o
jsonpath=$'{.items[0].metadata.name}\n')" \
    -- /usr/bin/dockerregistry -prune=delete
```

出力サンプル

```
$ oc exec docker-registry-3-vhndw \
```

```
    -- /usr/bin/dockerregistry -prune=delete

Deleted 13374 blobs
Freed up 2.835 GiB of disk space
```

7. レジストリーを読み取り書き込みモードに戻す: プルーニングの終了後は、以下を実行してレジストリーを読み取り書き込みモードに戻すことができます。

```
$ oc set env -n default dc/docker-registry
REGISTRY_STORAGE_MAINTENANCE_READONLY-
```

19.8. CRON ジョブのプルーニング

重要

cron ジョブについては、現時点で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 の実稼働環境でのサービスレベルアグリーメント (SLA) ではサポートされていないため、Red Hat では実稼働環境での使用を推奨していません。テクノロジープレビューの機能は、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についての詳細は、https://access.redhat.com/support/offerings/techpreview/ を参照してください。

cron ジョブは正常なジョブのプルーニングを実行できますが、失敗したジョブを適切に処理しない可能性があります。そのため、クラスター管理者は定期的な ジョブのクリーンアップ を手動で実行する必要があります。また、信頼できるユーザーの小規模なグループに cron ジョブへの アクセスを制限 し、cron ジョブでジョブや Pod が作成され過ぎないように適切な クォータ を設定することも推奨されます。

第20章 カスタムリソースによる KUBERNETES API の拡張

20.1. KUBERNETES カスタムリソース定義

Kubernetes API では、リソースは特定の種類の API オブジェクトのコレクションを保管するエンドポイントです。たとえば、ビルトインされた Pod リソースには Pod オブジェクトのコレクションが含まれます。

カスタムリソースは、Kubernetes API を拡張するか、またはプロジェクトまたはクラスターに独自の API を導入することを可能にするオブジェクトです。

カスタムリソース定義 (CRD) ファイルは、独自のオブジェクトの種類を定義し、API サーバーがライフサイクル全体を処理できるようにします。CRD をクラスターにデプロイすると、Kubernetes API サーバーは指定されたカスタムリソースを提供し始めます。

新規のカスタムリソース定義 (CRD) の作成時に、Kubernetes API サーバーは、クラスター全体または単一プロジェクト (namespace) でアクセスできる新規 RESTful リソースパスを作成することによって応答します。既存のビルトインオブジェクトの場合のように、プロジェクトを削除すると、そのプロジェクトのすべてのカスタムオブジェクトが削除されます。

CRD へのアクセスをユーザーに付与する場合、クラスターロールの集計により、管理、編集、または表示のデフォルトクラスターロールを持つユーザーにアクセスを付与します。また、クラスターロールの集計により、カスタムポリシールールをこれらのクラスターロールに挿入することができます。この動作は、これがビルトインリソースであるかのように新規リソースをクラスターの RBAC ポリシーに統合します。

注記

クラスター管理者のみが CRD を作成できますが、読み取りと書き込みのパーミッションがある場合には、CRD からオブジェクトを作成できます。

20.2. カスタムリソース定義の作成

カスタムオブジェクトを作成するには、まずカスタムリソース定義 (CRD) を作成する必要があります。

注記

クラスター管理者のみが CRD を作成できます。

手順

CRD を作成するには、以下を実行します。

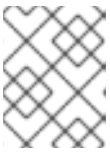
1. 以下の例のようなフィールドを含む YAML ファイルを作成します。

カスタムリソース定義の YAML ファイルの例

```
apiVersion: apiextensions.k8s.io/v1beta1 (1)
kind: CustomResourceDefinition
metadata:
  name: crontabs.stable.example.com (2)
spec:
  group: stable.example.com (3)
```

```
version: v1 4
scope: Namespaced 5
names:
  plural: crontabs 6
  singular: crontab 7
  kind: CronTab 8
  shortNames:
  - ct 9
```

1. **apiextensions.k8s.io/v1beta1** API を使用します。
2. 定義の名前を指定します。これは **group** および **plural** フィールドの値を使用する <plural-name><group> 形式である必要があります。
3. API のグループ名を指定します。API グループは、論理的に関連付けられるオブジェクトのコレクションです。たとえば、**Job** または **ScheduledJob** などのすべてのバッチオブジェクトはバッチ API グループ (batch.api.example.com など) である可能性があります。組織の完全修飾ドメイン名を使用することが奨励されます。
4. URL で使用されるバージョン名を指定します。それぞれの API グループは複数バージョンで存在させることができます。たとえば、**v1alpha**、**v1beta**、**v1** などが使用されます。
5. カスタムオブジェクトがクラスター (**Cluster**) の 1 つのプロジェクト (**Namespaced**) またはすべてのプロジェクトで利用可能であるかどうかを指定します。
6. URL で使用される複数形の名前を指定します。**plural** フィールドは API URL のリソースと同じになります。
7. CLI および表示用にエイリアスとして使用される単数形の名前を指定します。
8. 作成できるオブジェクトの種類を指定します。タイプは CamelCase にすることができます。
9. CLI でリソースに一致する短い文字列を指定します。



注記

デフォルトで、カスタムリソース定義のスコープはクラスターに設定され、すべてのプロジェクトで利用可能です。

2. オブジェクトを作成します。

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

新規の RESTful API エンドポイントは以下のように作成されます。

```
/apis/<spec:group>/<spec:version>/<scope>/*/<names-plural>/...
```

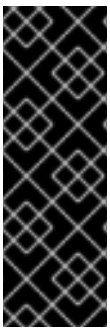
たとえば、サンプルファイルを使用すると、以下のエンドポイントが作成されます。

```
/apis/stable.example.com/v1/namespaces/*/crontabs/...
```

このエンドポイント URL を使用してカスタムオブジェクトを作成し、管理できます。オブジェクトの種類は、作成したカスタムリソース定義オブジェクトの **spec.kind** フィールドに基づいています。

20.3. カスタムリソース定義のクラスターロールの作成

クラススコープのカスタムリソース定義 (CRD) の作成後に、これに対してパーミッションを付与できます。管理、編集および表示の デフォルトクラスターロール を使用する場合、これらのルールにクラスターロールの集計を利用できます。



重要

これらのロールのいずれかにパーミッションを付与する際は、明示的に付与する必要があります。より多くのパーミッションを持つロールはより少ないパーミッションを持つロールからルールを継承しません。ルールをあるロールに割り当てる場合、寄り多くのパーミッションを持つロールにもその動詞を割り当てる必要もあります。たとえば、get crontabs パーミッションを表示ロールに付与する場合、これを編集および管理ロールにも付与する必要があります。admin または edit ロールは通常、プロジェクトテンプレート でプロジェクトを作成したユーザーに割り当てられます。

前提条件

- CRD を作成します。

手順

1. CRD のクラスターロール定義ファイルを作成します。クラスターロール定義は、各クラスターロールに適用されるルールが含まれる YAML ファイルです。OpenShift Container Platform Controller はデフォルトクラスターロールに指定するルールを追加します。

カスタムロール定義の YAML ファイルの例

```
apiVersion: rbac.authorization.k8s.io/v1 (1)
kind: ClusterRole
items:
 - metadata:
    name: aggregate-cron-tabs-admin-edit (2)
    labels:
      rbac.authorization.k8s.io/aggregate-to-admin: "true" (3)
      rbac.authorization.k8s.io/aggregate-to-edit: "true" (4)
  rules:
    - apiGroups: ["stable.example.com"] (5)
      resources: ["crontabs"] (6)
      verbs: ["get", "list", "watch", "create",
              "update", "patch", "delete", "deletecollection"] (7)
 - metadata:
    name: aggregate-cron-tabs-view (8)
    labels:
      # Add these permissions to the "view" default role.
      rbac.authorization.k8s.io/aggregate-to-view: "true" (9)
      rbac.authorization.k8s.io/aggregate-to-cluster-reader: "true" (10)
  rules:
```

```
  - apiGroups: ["stable.example.com"] 11
    resources: ["crontabs"] 12
    verbs: ["get", "list", "watch"] 13
```

1. **apiextensions.k8s.io/v1beta1** API を使用します。
2. 8 定義の名前を指定します。
3. パーミッションを管理のデフォルトロールに付与するためにこのラベルを指定します。
4. パーミッションを編集のデフォルトロールに付与するためにこのラベルを指定します。
5. 11 CRD のグループ名を指定します
6. 12 これらのルールが適用される CRD の複数形の名前を指定します。
7. 13 ロールに付与されるパーミッションを表す 動詞 を指定します。たとえば、**admin** および **edit** ロールに読み取りおよび書き込みパーミッションを適用し、**view** ロールに読み取りパーミッションのみを適用します。
9. パーミッションを **view** のデフォルトロールに付与するためにこのラベルを指定します。
10. パーミッションを **cluster-reader** のデフォルトロールに付与するためにこのラベルを指定します。

2. クラスターロールを作成します。

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

20.4. CRD からのカスタムオブジェクトの作成

カスタムリソース定義 (CRD) オブジェクトの作成後に、その仕様を使用するカスタムオブジェクトを作成できます。

カスタムオブジェクトには、任意の JSON コードを含むカスタムフィールドを含めることができます。

前提条件

- CRD を作成します。

手順

1. カスタムオブジェクトの YAML 定義を作成します。以下の定義例では、**cronSpec** と **image** のカスタムフィールドが **CronTab** タイプのカスタムオブジェクトに設定されます。このタイプは、カスタムリソース定義オブジェクトの **spec.kind** フィールドから取得します。

   カスタムオブジェクトの YAML ファイルの例

```
apiVersion: "stable.example.com/v1" 1
kind: CronTab 2
metadata:
  name: my-new-cron-object 3
  finalizers: 4
  - finalizer.stable.example.com
```

```
spec: 5
  cronSpec: "* * * * /5"
  image: my-awesome-cron-image
```

1. カスタムリソース定義からグループ名および API バージョン (名前/バージョン) を指定します。
2. カスタムリソース定義のタイプを指定します。
3. オブジェクトの名前を指定します。
4. オブジェクトの ファイナライザー を指定します (ある場合)。ファイナライザーは、コントローラーがオブジェクトの削除前に完了する必要のある条件を実装できるようにします。
5. オブジェクトのタイプに固有の条件を指定します。

2. オブジェクトファイルの作成後に、オブジェクトを作成します。

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

20.5. カスタムオブジェクトの管理

オブジェクトを作成した後には、カスタムリソースを管理できます。

前提条件

- カスタムリソース定義 (CRD) を作成します。
- CRD からオブジェクトを作成します。

手順

1. 特定の種類のカスタムリソースについての情報を取得するには、以下を入力します。

```
oc get <kind>
```

以下に例を示します。

```
oc get crontab

NAME                 KIND
my-new-cron-object   CronTab.v1.stable.example.com
```

リソース名では大文字と小文字が区別されず、CRD で定義される単数形または複数形のいずれか、および任意の短縮名を指定できることに注意してください。以下に例を示します。

```
oc get crontabs
oc get crontab
oc get ct
```

2. カスタムリソースの未加工の YAML データも確認することができます。

```
oc get <kind> -o yaml
```

```
oc get ct -o yaml

apiVersion: v1
items:
- apiVersion: stable.example.com/v1
  kind: CronTab
  metadata:
    clusterName: ""
    creationTimestamp: 2017-05-31T12:56:35Z
    deletionGracePeriodSeconds: null
    deletionTimestamp: null
    name: my-new-cron-object
    namespace: default
    resourceVersion: "285"
    selfLink: /apis/stable.example.com/v1/namespaces/default/crontabs/my-new-cron-object
    uid: 9423255b-4600-11e7-af6a-28d2447dc82b
  spec:
    cronSpec: '* * * * /5' 1
    image: my-awesome-cron-image 2
```

1 2 オブジェクトの作成に使用した YAML からのカスタムデータが表示されます。

第21章 ガベージコレクション

21.1. 概要

OpenShift Container Platform ノードは、2 種類のガベージコレクションを実行します。

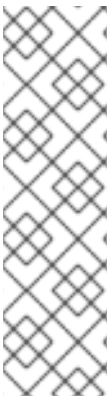
- コンテナーのガベージコレクション: 終了したコンテナーを削除します。デフォルトでは有効です。
- イメージのガベージコレクション: 実行中の Pod によって参照されていないイメージを削除します。デフォルトでは有効になっていません。

21.2. コンテナーのガベージコレクション

コンテナーのガベージコレクションはデフォルトで有効にされ、エビクションのしきい値に達すると自動的に実行されます。ノードは Pod のコンテナーを API からアクセス可能な状態にしようとします。Pod が削除された場合、コンテナーも削除されます。コンテナーは Pod が削除されておらず、エビクションのしきい値に達していない限り保持されます。ノードがディスク不足 (disk pressure) の状態にある場合、コンテナーが削除され、それらのログは **oc logs** でアクセスできなくなります。

コンテナーのガベージコレクションのポリシーは 3 つのノード設定に基づいています。

設定	説明
minimum-container-ttl-duration	コンテナーがガベージコレクションの対象となるのに必要な最小の年数です。デフォルトは 0 です。制限なしにするには 0 を使用します。この設定の値は、時間の h、分の m、秒の s などの単位の接尾辞を使用して指定することができます。
maximum-dead-containers-per-container	コンテナーごとに保持する古いインスタンス数。デフォルトは 1 です。
maximum-dead-containers	ノードにある実行されないコンテナーの合計の最大数です。デフォルトは、無制限を意味する -1 です。

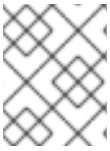


注記

競合が生じる場合、**maximum-dead-containers** 設定は **maximum-dead-containers-per-container** 設定よりも優先されます。たとえば、**maximum-dead-containers-per-container** の数を保持することでコンテナーの合計数が **maximum-dead-containers** より大きくなる場合、最も古いコンテナーが削除され、**maximum-dead-containers** の制限が満たされるようにします。

ノードが実行されていないコンテナーを削除すると、それらのコンテナーの内部にあるすべてのファイルも削除されます。そのノードで作成されたコンテナーに対してのみガベージコレクションが実行されます。

デフォルト設定を使用しない場合は、適切な ノード設定マップ の **kubeletArguments** セクションで、これらの設定の値を指定できます。このセクションがまだ存在しない場合は追加します。



注記

コンテナーのガベージコレクションは、これらのパラメーターがノード設定マップに存在しない場合にデフォルト値を使用して実行されます。

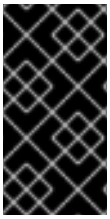
コンテナーのガベージコレクション設定

```
kubeletArguments:
  minimum-container-ttl-duration:
    - "10s"
  maximum-dead-containers-per-container:
    - "2"
  maximum-dead-containers:
    - "240"
```

21.2.1. 削除するコンテナーの検出

ガベージコレクターループの各スピンは、次の手順を実行します。

1. 利用可能なコンテナーの一覧を取得します。
2. 実行中であるか、または **minimum-container-ttl-duration** パラメーターよりも長く存続していないすべてのコンテナーをフィルターで除外します。存続していないコンテナーは、終了、停止、または終了した状態になる可能性があります。
3. 残りのすべてのコンテナーを Pod およびイメージ名のメンバーシップに基づいて同等のクラスに分類します。
4. 特定されないコンテナー (kubelet で管理されているコンテナーであるが、それらの名前の形式が正しくないコンテナー) をすべて削除します。
5. **maximum-dead-containers-per-container** パラメーターよりも多くのコンテナーが含まれるそれぞれのクラスについて、そのクラスのコンテナーを作成時間で並び替えます。
6. **maximum-dead-containers-per-container** パラメーターの条件が満たされるまで、コンテナーを最も古いものから順に削除し始めます。
7. 依然として **maximum-dead-containers** パラメーターよりも多くのコンテナーが一覧にある場合、コレクターは各クラスのコンテナーの削除を開始し、それぞれのクラスにあるコンテナー数がクラスあたりのコンテナーの平均数、または **<all_remaining_containers>/<number_of_classes>** よりも大きくならないようにします。
8. これがまだ十分でない場合は、コレクターは一覧のすべてのコンテナーを分類し、**maximum-dead-containers** の基準が満たされるまで、コンテナーを最も古いものから順番に削除し始めます。



重要

ニーズに合わせてデフォルト設定を更新します。

ガベージコレクションは、関連付けられている Pod のないコンテナーのみを削除します。

21.3. イメージのガベージコレクション

イメージのガベージコレクションでは、ノードの **cAdvisor** によって報告されるディスク使用量に基づいて、ノードから削除するイメージを決定します。この場合、以下の設定が考慮に入れられます。

設定	説明
image-gc-high-threshold	イメージのガベージコレクションをトリガーするディスク使用量のパーセント (整数で表される) です。
image-gc-low-threshold	イメージのガベージコレクションが解放しようとするディスク使用量のパーセント (整数で表される) です。

イメージのガベージコレクションを有効にするには、適切な ノード設定マップ の **kubeletArguments** セクションで、これらの設定の値を指定します。このセクションがまだ存在しない場合は追加します。

注記

イメージのガベージコレクションは、これらのパラメーターがノード設定マップに存在しない場合にデフォルト値を使用して実行されます。

イメージのガベージコレクション設定

```
kubeletArguments:
  image-gc-high-threshold:
    - "85"
  image-gc-low-threshold:
    - "80"
```

21.3.1. 削除するイメージの検出

以下の 2 つのイメージ一覧がそれぞれのガベージコレクターの実行で取得されます。

1. 1 つ以上の Pod で現在実行されているイメージの一覧
2. ホストで利用可能なイメージの一覧

新規コンテナーの実行時に新規のイメージが表示されます。すべてのイメージにはタイムスタンプのマークが付けられます。イメージが実行中 (上記の最初の一覧) か、または新規に検出されている (上記の 2 番目の一覧) 場合、これには現在の時間のマークが付けられます。残りのイメージには以前のタイムスタンプのマークがすでに付けられています。すべてのイメージはタイムスタンプで並び替えられます。

コレクションが開始されると、停止条件を満たすまでイメージが最も古いものから順番に削除されます。

第22章 ノードリソースの割り当て

22.1. ノードリソースの割り当て目的

より信頼性の高いスケジューリングを実現し、ノードにおけるリソースのオーバーコミットを最小限にするために、kubelet、kube-proxy およびコンテナーエンジンなどの基礎となる ノードのコンポーネント に使用される CPU およびメモリーリソースの一部を予約します。予約するリソースは、**sshd**、**NetworkManager** などの残りのシステムコンポーネントによっても使用されます。予約するリソースを指定して、スケジューラーに、ノードが Pod で使用できる残りのメモリーおよび CPU リソースに関する詳細を提供します。

22.2. 割り当てられたリソースのノードの設定

リソースは、**system-reserved** ノード設定を設定して OpenShift Container Platform のノードコンポーネントおよびシステムコンポーネント用に予約されます。

OpenShift Container Platform は **kube-reserved** 設定を使用しません。Kubernetes および Kubernetes 環境を提供する一部のクラウドベンダーのドキュメントでは、**kube-reserved** の設定が提案されている場合があります。この情報は OpenShift Container Platform クラスターには適用されません。

リソース制限を使用してクラスターを調整し、エビクションを使用して制限を適用する場合は注意が必要です。**system-reserved** 制限を適用すると、メモリーリソースが不足するときに、重要なシステムサービスが CPU 時間を受信したり、重要なシステムサービスを終了したりするのを防ぐことができます。

ほとんどの場合、リソースの割り当ての調整は、調整を行ってから、本番環境のようなワークロードでクラスターのパフォーマンスを監視することによって実行されます。このプロセスは、クラスターが安定し、サービスレベルアグリーメントを満たすまで繰り返されます。

これらの設定の影響については、割り当てられるリソースの計算 を参照してください。

設定	説明
kube-reserved	この設定は OpenShift Container Platform では使用されません。予約する予定の CPU とメモリーリソースを **system-reserved** 設定に追加します。
system-reserved	ノードコンポーネントおよびシステムコンポーネント用に予約されたリソースです。デフォルトは none です。

以下のコマンドを実行して、**lscgroup** などのツールで **system-reserved** により制御されるサービスを表示します。

```
# yum install libcgroup-tools
```

```
$ lscgroup memory:/system.slice
```

<resource_type>=<resource_quantity> ペアのセットを追加して、ノード設定マップ の **kubeletArguments** セクションのリソースを予約します。たとえば、**cpu=500m,memory=1Gi** は CPU の 500 ミリコアおよび 1 ギガバイトのメモリーを予約します。

例22.1 ノードの割り当て可能なリソースの設定

```
kubeletArguments:
  system-reserved:
    - "cpu=500m,memory=1Gi"
```

system-reserved フィールドが存在しない場合は追加します。

注記

node-config.yaml ファイルを直接編集しないでください。

これらの設定に適切な値を決定するには、ノード要約 API を使用して、ノードのリソース使用量を表示します。詳細は、ノードによって報告されるシステムリソース を参照してください。

system-reserved の設定後に、以下を実行します。

- ノードのメモリー使用量をモニターして、高基準値を確認します。

  ```
  $ ps aux | grep <service-name>
  ```

  以下に例を示します。

  ```
  $ ps aux | grep atomic-openshift-node

  USER       PID   %CPU  %MEM  VSZ    RSS  TTY    STAT  START  TIME  COMMAND
  root       11089 11.5  0.3   112712  996  pts/1  R+    16:23  0:00  grep --color=auto atomic-
  openshift-node
  ```

  この値が **system-reserved** マークに近い場合は、**system-reserved** 値を増やすことができます。

- 以下のコマンドを実行して、**cgget** などのツールで、システムサービスのメモリー使用量を監視します。

  ```
  # yum install libcgroup-tools
  ```

  ```
  $ cgget -g memory  /system.slice | grep memory.usage_in_bytes
  ```

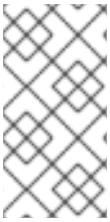
  この値が **system-reserved** マークに近い場合は、**system-reserved** 値を増やすことができます。

- OpenShift Container Platform クラスターローダー を使用して、さまざまなクラスター状態でのデプロイメントのパフォーマンスメトリクスを測定します。

22.3. 割り当てられたリソースの計算

割り当てられたリソースの量は、以下の数式に基づいて計算されます。

```
[Allocatable] = [Node Capacity] - [system-reserved] - [Hard-Eviction-Thresholds]
```



注記

Allocatable の値がノードレベルで Pod に対して適用されるために、**Hard-Eviction-Thresholds** を Allocatable から差し引くと、システムの信頼性が強化されます。**experimental-allocatable-ignore-eviction** 設定は、レガシー動作を保持するために利用できますが、今後のリリースでは非推奨となります。

[Allocatable] が負の値の場合、これは **0** に設定されます。

22.4. ノードの割り当て可能なリソースおよび容量の表示

ノードの現在の容量と割り当て可能なリソースを表示するには、以下のコマンドを実行します。

```
$ oc get node/<node_name> -o yaml
```

以下の部分的な出力では、割り当て可能な値は容量よりも少なくなります。この違いは想定されており、**system-reserved** の **cpu=500m,memory=1Gi** リソースの割り当てに一致します。

```
status:
...
  allocatable:
    cpu: "3500m"
    memory: 6857952Ki
    pods: "110"
  capacity:
    cpu: "4"
    memory: 8010948Ki
    pods: "110"
...
```

スケジューラーは、**allocatable** の値を使用して、ノードが Pod スケジューリングの候補であるかどうかを判別します。

22.5. ノードによって報告されるシステムリソース

各ノードはコンテナーランタイムおよび kubelet によって利用されるシステムリソースについて報告します。**system-reserved** の設定を簡素化するには、ノード要約 API を使用してノードのリソース使用状況を表示します。ノードの要約は **<master>/api/v1/nodes/<node>/proxy/stats/summary** で利用できます。

たとえば、**cluster.node22** ノードからリソースにアクセスするには、以下のコマンドを実行します。

```
$ curl <certificate details> https://<master>/api/v1/nodes/cluster.node22/proxy/stats/summary
```

応答には、以下のような情報が含まれます。

```
{
  "node": {
    "nodeName": "cluster.node22",
    "systemContainers": [
      {
        "cpu": {
          "usageCoreNanoSeconds": 929684480915,
```

```
            "usageNanoCores": 190998084
          },
          "memory": {
            "rssBytes": 176726016,
            "usageBytes": 1397895168,
            "workingSetBytes": 1050509312
          },
          "name": "kubelet"
        },
        {
          "cpu": {
            "usageCoreNanoSeconds": 128521955903,
            "usageNanoCores": 5928600
          },
          "memory": {
            "rssBytes": 35958784,
            "usageBytes": 129671168,
            "workingSetBytes": 102416384
          },
          "name": "runtime"
        }
      ]
    }
}
```


証明書の詳細については、REST API Overview を参照してください。

22.6. ノードの実施

ノードは、Pod が設定された割り当て可能な値に基づいて消費できるリソースの合計量を制限できます。この機能は、Pod がシステムサービス (コンテナーランタイム、ノードエージェントなど) で必要とされる CPU およびメモリーリソースを使用することを防ぎ、ノードの信頼性を大幅に強化します。ノードの信頼性を強化するために、管理者はリソースの使用についてのターゲットに基づいてリソースを確保する必要があります。

ノードは、QoS (Quality of Service) を実施する新規の cgroup 階層を使用してリソースの制約を実施します。すべての Pod は、システムデーモンから切り離された専用の cgroup 階層で起動されます。

ノードの実施を設定するには、適切な ノード設定マップ で、以下のパラメーターを使用します。

例22.2 ノードの **cgroup** 設定

```
kubeletArguments:
  cgroups-per-qos:
    - "true" 1
  cgroup-driver:
    - "systemd" 2
  enforce-node-allocatable:
    - "pods" 3
```

1 それぞれの QoS (Quality of Service) の cgroup 階層を有効化または無効化します。cgroups はノードによって管理されます。この設定の変更にはノードの完全なドレイン (解放) が必要になります。ノード割り当て可能リソース制約をノードが実施できるようにするには、このフラグを **true** にする必要があります。デフォルト値は **true** です。Red Hat では、この値を変更することを推奨していません。

2 cgroup 階層を管理するためにノードによって使用される cgroup ドライバーです。この値はコンテナーランタイムに関連付けられたドライバーに一致する必要があります。有効な値は

3 ノードがノードのリソース制約を実施するスコープのコンマ区切りの一覧です。デフォルト値は **pods** です。Red Hat は **Pod** のみをサポートします。

管理者は Guaranteed QoS (Quality of Service) のある Pod と同様にシステムデーモンを処理する必要があります。システムデーモンは、境界となる制御グループ内でバーストする可能性があり、この動作はクラスターのデプロイメントの一部として管理される必要があります。割り当て済みのリソースのノードの設定 セクションで説明されているように、**system-reserved** でリソースを指定して、システムデーモンの CPU およびメモリーリソースを予約します。

設定されている cgroup ドライバーを表示するには、以下のコマンドを実行します。

```
$ systemctl status atomic-openshift-node -l | grep cgroup-driver=
```

出力には、以下のような応答が含まれます。

```
--cgroup-driver=systemd
```

cgroup ドライバーの管理およびトラブルシューティングに関する詳細は、コントロールグループ (Cgroups) の概要 を参照してください。

22.7. エビクションしきい値

ノードがメモリー不足の状態にある場合、ノード全体、およびノードで実行されているすべての Pod に影響が及ぶ可能性があります。システムデーモンがメモリーの予約量を超える量を使用する場合、メモリー不足のイベントが発生し、ノード全体とノードで実行されているすべての Pod に影響を与える可能性があります。システムのメモリー不足のイベントを防止するか、またはそれが発生する可能性を軽減するために、ノードは リソース不足の処理 (out of resource handling) を行います。

22.8. 関連リソース

- オーバーコミット
- Out of Resource (リソース不足) エラーの処理
- 制限範囲の設定

第23章 オーバーコミット

23.1. 概要

コンテナーは、コンピュートリソース要求および制限 を指定することができます。要求はコンテナー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。一方、制限はノードで消費できるコンピュート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トリソース使用の最適化を試行します。これは Pod のコンピュート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。

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。 これは、保証されるパフォーマンスとキャパシティーのトレードオフが許容される開発環境において役立ちます。

23.2. 要求および制限

各コンピュートリソースについて、コンテナー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。コンテナー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。コンテナーは、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、コンピュートリソースのタイプによって異なります。コンテナーが要求または制限を指定しない場合、コンテナー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、コンテナー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ーに最低レベルの QoS (Quality of Service) が設定されます。

23.2.1. Buffer Chunk Limit の調整

Fluentd ロガーが多数のログを処理できない場合、メモリーの使用量を減らし、データ損失を防ぐためにファイルバッファーリングに切り換える必要があります。

Fluentd **buffer_chunk_limit** は、デフォルト値が **8m** の環境変数 **BUFFER_SIZE_LIMIT** によって決定されます。出力ごとのファイルのバッファーサイズは、デフォルト値が **256Mi** の環境変数 **FILE_BUFFER_LIMIT** によって決定されます。永続的なボリュームサイズは、**FILE_BUFFER_LIMIT** に出力を乗算した結果よりも大きくなければなりません。

Fluentd および Mux Pod では、永続ボリューム **/var/lib/fluentd** は PVC または hostmount などによって作成される必要があります。その領域はファイルバッファーに使用されます。

buffer_type および **buffer_path** は、以下のように Fluentd 設定ファイルで設定されます。

```
$ egrep "buffer_type|buffer_path" *.conf
output-es-config.conf:
  buffer_type file
  buffer_path `/var/lib/fluentd/buffer-output-es-config`
output-es-ops-config.conf:
  buffer_type file
  buffer_path `/var/lib/fluentd/buffer-output-es-ops-config`
filter-pre-mux-client.conf:
  buffer_type file
  buffer_path `/var/lib/fluentd/buffer-mux-client`
```

Fluentd **buffer_queue_limit** は変数 **BUFFER_QUEUE_LIMIT** の値です。この値はデフォルトで **32** になります。

環境変数 **BUFFER_QUEUE_LIMIT** は **(FILE_BUFFER_LIMIT / (number_of_outputs * BUFFER_SIZE_LIMIT))** として計算されます。

BUFFER_QUEUE_LIMIT 変数にデフォルトの値のセットが含まれる場合、以下のようになります。

- **FILE_BUFFER_LIMIT = 256Mi**
- **number_of_outputs = 1**
- **BUFFER_SIZE_LIMIT = 8Mi**

buffer_queue_limit の値は **32** になります。**buffer_queue_limit** を変更するには、**FILE_BUFFER_LIMIT** の値を変更する必要があります。

この数式では、**number_of_outputs** は、すべてのログが単一リソースに送信され、追加のリソースごとに **1** つずつ増分する場合に **1** になります。たとえば、**number_of_outputs** の値は以下のようになります。

- **1** - すべてのログが単一の ElasticSearch Pod に送信される場合
- **2** - アプリケーションログが ElasticSearch Pod に送信され、運用ログが別の ElasticSearch Pod に送信される場合
- **4** - アプリケーションログが ElasticSearch Pod に送信され、運用ログが別の ElasticSearch Pod に送信される場合で、それらがどちらも他の Fluentd インスタンスに転送される場合

23.3. コンピュートリソース

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。

23.3.1. CPU

コンテナーには要求する CPU の量が保証され、さらにコンテナー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。複数のコンテナーが追加の CPU の使用を試行する場合、CPU 時間が各コンテナーで要求される CPU の量に基づいて分配されます。

たとえば、あるコンテナーが 500m の CPU 時間を要求し、別のコンテナーが 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 の比率でコンテナー間で分配されます。コンテナー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実施されます。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。

23.3.2. メモリー

コンテナーには要求するメモリー量が保証されます。コンテナー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。

コンテナー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。コンテナー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。

23.3.3. 一時ストレージ

注記

このトピックは、一時ストレージのテクノロジープレビューを有効にした場合にのみ適用されます。この機能はデフォルトでは無効になっています。有効にすると、OpenShift Container Platform クラスターは一時ストレージを使用して、クラスターが破棄された後に永続される必要のない情報を保存します。この機能を有効にするには、configuring for ephemeral storage を参照してください。

コンテナーには要求する一時ストレージの量が保証されます。コンテナーは要求したよりも多くの一時ストレージを使用できますが、いったん要求した量を超えた場合には、利用可能な一時ディスク領域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。

コンテナーが要求した量よりも少ない一時ストレージを使用する場合、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ローカルの一時ストレージ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。コンテナーが一時ストレージ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。

23.4. QOS (QUALITY OF SERVICE) クラス

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。

各コンピュートリソースについて、コンテナーは 3 つの QoS クラスに分類されます (優先順位は降順)。

表23.1 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、コンテナーは **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、コンテナーは **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、コンテナーは **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ーが最初に強制終了されます。

- **Guaranteed** コンテナーは優先順位が最も高いコンテナーとして見なされ、保証されます。 強制終了されるのは、これらのコンテナーで制限を超えるか、または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ーが他にない場合のみです。

- システム不足の状態にある **Burstable** コンテナーは、制限を超過し、**BestEffort** コンテナー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** コンテナーは優先順位の最も低いコンテナーとして処理されます。これらのコンテナーのプロセスは、システムがメモリーー不足になると最初に強制終了されます。

23.5. マスターでのオーバーコミットの設定

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、コンテナーに 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、コンテナー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。

OpenShift Container Platform 管理者がオーバーコミットのレベルを制御し、ノードのコンテナー密度を管理する必要がある場合、開発者コンテナーで設定された要求と制限の比率を上書きするようマスターを設定することができます。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ーの制限と要求を調整することができます。

これを実行するには、以下の例にあるように **master-config.yaml** で **ClusterResourceOverride** 受付コントローラーを設定することが必要です (既存の設定ツリーが存在する場合はこれを再利用するか、または必要に応じて存在しない要素を導入します)。

```
admissionConfig:
  pluginConfig:
    ClusterResourceOverride: 1
      configuration:
        apiVersion: v1
        kind: ClusterResourceOverrideConfig
        memoryRequestToLimitPercent: 25 2
        cpuRequestToLimitPercent: 25 3
        limitCPUToMemoryPercent: 200 4
```

1. これはプラグイン名です。大文字/小文字の区別が必要であり、プラグインの完全に一致する名前以外はすべて無視されます。

2. (オプション、1-100) コンテナー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このパーセンテージに対応して上書きされます。

3. (オプション、1-100) コンテナーの CPU 制限が指定されているか、またはデフォルトに設定されている場合、CPU 要求は制限のこのパーセンテージに対応して上書きされます。

4. (オプション、正の整数) コンテナー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 制限はメモリー制限のパーセンテージに対応して上書きされます。この場合、1Gi の RAM が 1 CPU コアと等しくなるように 100% スケーリングされます。この場合、1Gi の RAM が 1 CPU コアと等しくなる場合に 100 パーセントになります。 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)。

マスター設定の変更後は、マスターの再起動が必要になります。

制限がコンテナーに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ないことに注意してください。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 を使用して)、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。

また、上書き後も、コンテナーの制限および要求がプロジェクトのいずれかの LimitRange オブジェクトで依然として検証される必要があることにも注意してください。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 LimitRanges を注意して設定してください。

上書きが設定されている場合に、プロジェクトを編集し、以下のアノテーションを追加することで、上書き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の設定を上書きと切り離して実行できます)。

```
quota.openshift.io/cluster-resource-override-enabled: "false"
```

23.6. ノードでのオーバーコミットの設定

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。

23.6.1. Quality of Service (QoS) 層でのメモリー予約

experimental-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 OoS 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。

高い QOS レベル用にリソースを予約することで、リソース制限を持たない Pod が高い QoS レベルの Pod で要求されるリソースを侵害しないようにできます。

experimental-qos-reserved パラメーターを設定するには、適切な ノード設定マップ を編集します。

```
kubeletArguments:
  cgroups-per-qos:
  - true
  cgroup-driver:
  - 'systemd'
  cgroup-root:
  - '/'
  experimental-qos-reserved: (1)
  - 'memory=50%'
```

(1) Pod のリソース要求が QoS レベルでどのように予約されるかを指定します。

OpenShift Container Platform は、以下のように **experimental-qos-reserved** パラメーターを使用します。

- **experimental-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。

- **experimental-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。

- **experimental-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。

23.6.2. CPU 制限の実施

デフォルトで、ノードは Linux カーネルの CPU CFS クォータのサポートを使用して指定された CPU 制限を実施します。ノードで CPU 制限を実施する必要がない場合は、以下のパラメーターを含めるように、適切な ノード設定マップ を変更してその実施を無効にすることができます。

```
kubeletArguments:
  cpu-cfs-quota:
    - "false"
```

CPU 制限の実施が無効にされる場合、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。

- コンテナーが CPU の要求をする場合、これは Linux カーネルの CFS 共有によって引き続き実施されます。

- コンテナーが CPU の要求を明示的に指定しないものの、制限を指定する場合には、要求は指定された制限にデフォルトで設定され、Linux カーネルの CFS 共有で実施されます。

- コンテナーが CPU の要求と制限の両方を指定する場合、要求は Linux カーネルの CFS 共有で実施され、制限はノードに影響を与えません。

23.6.3. システムリソースのリソース予約

スケジューラー は、Pod 要求に基づいてノード上のすべての Pod に十分なリソースがあることを確認します。これは、ノード上のコンテナーの要求の合計がノード容量を上回らないことを確認します。これには、ノードで起動されたすべてのコンテナーが含まれますが、クラスターの範囲外で起動されたコンテナーやプロセスは含まれません。

ノード容量の一部を予約して、クラスターが機能できるようノードで実行する必要のあるシステムデーモン用に確保することが推奨されます (**sshd**、**docker** など)。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予約を行うことが推奨されます。

Pod 以外のプロセスのリソースを明示的に予約する必要がある場合、以下の 2 つの方法でこれを実行できます。

- 優先される方法として、スケジューリングに利用できるリソースを指定してノードリソースを割り当てることができます。詳細は、ノードリソースの割り当て を参照してください。

- または、クラスターによってノードでスケジュールされないように容量を予約するだけの **resource-reserver** Pod を作成することもできます。以下に例を示します。

  例23.1resource-reserver Pod の定義

```
apiVersion: v1
kind: Pod
metadata:
  name: resource-reserver
spec:
  containers:
  - name: sleep-forever
    image: gcr.io/google_containers/pause:0.8.0
    resources:
      limits:
        cpu: 100m (1)
        memory: 150Mi (2)
```

(1) クラスターに認識されないホストレベルのデーモン用にノード上で確保する CPU の量です。

(2) クラスターに認識されないホストレベルのデーモン用にノード上で確保するメモリーの量です。

定義は **resource-reserver.yaml** のようなファイルに保存し、ファイルを **/etc/origin/node/** または別の指定がある場合は **--config=<dir>** などのノード設定ディレクトリーに置くことができます。

さらに、ディレクトリーを適切な ノード設定マップ の **kubeletArguments.config** パラメーターで指定し、ノード設定ディレクターから定義を読み取るようにノードサーバーを設定します。

```
kubeletArguments:
  config:
    - "/etc/origin/node" (1)
```

(1) **--config=<dir>** が指定されている場合、ここでは **<dir>** を使用します。

resource-reserver.yaml ファイルが有効な状態でノードサーバーを起動すると、**sleep-forever** コンテナーも起動します。スケジューラーはノードの残りの容量も考慮し、クラスター Pod を配置する場所を適宜調整します。

resource-reserver Pod を削除するには、ノード設定ディレクトリーから **resource-reserver.yaml** ファイルを削除するか、またはこれを移動することができます。

23.6.4. カーネルの調整可能なフラグ

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はありません。

この動作を確認するために、ノードはカーネルに対し、常にメモリーのオーバーコミットを実行するように指示します。

```
$ sysctl -w vm.overcommit_memory=1
```

また、ノードはカーネルに対し、メモリーが不足する状況でもパニックにならないように指示します。その代わりに、カーネルの OOM killer は優先順位に基づいてプロセスを強制終了します。

```
$ sysctl -w 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。

23.6.5. swap メモリーの無効化

OpenShift Container Platform 3.9 では、Ansible ノードのインストールの一環として swap が無効になっています。swap の有効化はサポート対象外になりましたが、今後のリリースにおける swap の適切なサポートを現在評価しています。

重要

swap を有効にして実行すると、意図しない結果がもたらされます。swap が有効にされている場合、利用可能なメモリーの リソース不足の処理 (out of resource handling) のエビクションしきい値は、想定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。

第24章 OUT OF RESOURCE (リソース不足) エラーの処理

24.1. 概要

このトピックでは、OpenShift Container Platform がメモリー不足 (OOM) やディスク領域不足の状況を防ぐためのベストエフォートの取り組みについて説明します。

ノードは、利用可能なコンピュートリソースが少ない場合に安定性を維持する必要があります。これは、メモリーやディスクなどの圧縮不可能なリソースを扱う場合にとくに重要になります。どちらかのリソースが消費されると、ノードは不安定になります。

管理者は、設定可能な エビクションポリシー を使用して、ノードをプロアクティブに監視し、ノードでコンピュートリソースおよびメモリーリソースが不足する状況を防ぐことができます。

このトピックでは、OpenShift Container Platform がリソース不足の状況に対処する方法についての情報を提供し、シナリオ例 および 推奨される対策 について説明します。

- リソースの回収
- Pod のエビクション
- Pod のスケジューリング
- リソース不足および Out of Memory Killer

警告

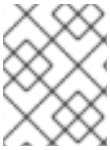
swap メモリーがノードに対して有効にされている場合、ノードはメモリー不足の状態かどうかを検出できません。

メモリーベースのエビクションを利用するには、オペレーターは swap を無効にする 必要があります。

24.2. エビクションポリシーの設定

エビクションポリシー により、ノードが利用可能なリソースが少ない状況で実行されている場合に 1 つ以上の Pod が失敗することを許可します。Pod の失敗により、ノードは必要なリソースを回収できます。

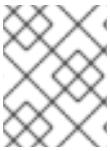
エビクションポリシーは、エビクショントリガーシグナル と、ノード設定ファイルまたは コマンドライン で設定される特定の エビクションしきい値 の組み合わせになります。エビクションは、ノードがしきい値を超える Pod に対して即時のアクションを実行する ハード エビクションか、またはノードがアクションを実行する前の猶予期間を許可する ソフト エビクションのどちらかになります。



注記

クラスターのノードを変更するには、ノード設定マップ を必要に応じて更新します。**node-config.yaml** ファイルは手動で変更しないようにしてください。

適切に設定されたエビクションポリシーを使用することで、ノードは、プロアクティブにモニターし、コンピュートリソースを完全に使い切る事態を防ぐことができます。



注記

ノードによる Pod の失敗が生じる場合、ノードは Pod のすべてのコンテナーを終了し、**PodPhase** は **Failed** に切り替わります。

ノードは、ディスクの不足状態を検出する際に **nodefs** および **imagefs** ファイルシステムのパーティションをサポートします。

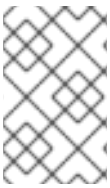
nodefs または **rootfs** は、ノードがローカルディスクボリューム、デーモンログ、emptyDir、および他のローカルストレージに使用するファイルシステムです。たとえば、**rootfs** は、**/** を提供するファイルシステムです。**rootfs** には、**openshift.local.volumes** (デフォルトは /var/lib/origin/openshift.local.volumes) が含まれます。

imagefs は、コンテナーランタイムがイメージおよび個別のコンテナーの書き込み可能な層を保存するために使用するファイルシステムです。エビクションのしきい値は、**imagefs** については 85% になります。**imagefs** ファイルシステムはランタイムによって異なり、Docker の場合は、コンテナーが使用するストレージドライバーによって異なります。

- Docker の場合:
  - **devicemapper** ストレージドライバーを使用する場合、**imagefs** はシンプールになります。
    Docker デーモンに **--storage-opt dm.basesize** フラグを設定して、コンテナーの読み取り/書き込み層を制限できます。

    ```
    $ sudo dockerd --storage-opt dm.basesize=50G
    ```

  - **overlay2** ストレージドライバーを使用している場合、**imagefs** は /var/lib/docker/overlay2 が含まれるファイルシステムになります。
- オーバーレイドライバーを使用する CRI-O の場合、**imagefs** は、デフォルトで /var/lib/containers/storage になります。



注記

ローカルストレージの分離 (一時ストレージ) を使用せず、XFS クォータ (volumeConfig) を使用しない場合、Pod でローカルディスクの使用を制限することはできません。

24.2.1. ノード設定を使用したポリシーの作成

エビクションポリシーを設定するには、適切な ノード設定ファイル を編集して、**eviction-hard** または **eviction-soft** パラメーターの下にエビクションしきい値を指定します。

以下のサンプルは、エビクションしきい値を示しています。

ハードエビクションのノード設定ファイルのサンプル

```
kubeletArguments:
  eviction-hard: 1
  - memory.available<100Mi 2
```

```
- nodefs.available<10%
- nodefs.inodesFree<5%
- imagefs.available<15%
- imagefs.inodesFree<10%
```

1 エビクションのタイプ: ハードエビクション にこのパラメーターを使用します。

2 特定のエビクショントリガーシグナルに基づくエビクションのしきい値です。

注記

inodesFree パラメーターのパーセント値を指定する必要があります。他のパラメーターのパーセンテージまたは数値を指定できます。

ソフトエビクションのノード設定ファイルのサンプル

```
kubeletArguments:
  eviction-soft: 1
  - memory.available<100Mi 2
  - nodefs.available<10%
  - nodefs.inodesFree<5%
  - imagefs.available<15%
  - imagefs.inodesFree<10%
  eviction-soft-grace-period: 3
  - memory.available=1m30s
  - nodefs.available=1m30s
  - nodefs.inodesFree=1m30s
  - imagefs.available=1m30s
  - imagefs.inodesFree=1m30s
```

1 エビクションのタイプ: ソフトエビクション にこのパラメーターを使用します。

2 特定のエビクショントリガーシグナルに基づくエビクションのしきい値です。

3 ソフトエビクションの猶予期間です。パフォーマンスを最適化するためにデフォルト値のままにします。

変更を有効にするために OpenShift Container Platform サービスを再起動します。

```
# systemctl restart atomic-openshift-node
```

24.2.2. エビクションシグナルについて

以下の表にあるシグナルのいずれかに基づいてエビクションの意思決定をトリガーするようノードを設定することができます。しきい値と共に、エビクションシグナルを エビクションのしきい値 に追加します。

シグナルを表示するには、以下を実行します。

```
curl <certificate details> \
  https://<master>/api/v1/nodes/<node>/proxy/stats/summary
```

表24.1 サポートされるエビクションシグナル

ノードの状態	エビクションシグナル	値	説明
MemoryPressure	**memory.available**	**memory.available** = **node.status.capacity[memory]** - **node.stats.memory.workingSet**	ノードの利用可能なメモリーがエビクションしきい値を超えている。
DiskPressure	**nodefs.available**	**nodefs.available** = **node.stats.fs.available**	ノードの root ファイルシステムまたはイメージファイルシステムのいずれかで、利用可能なディスク領域がエビクションしきい値を超えている。
	nodefs.inodesFree	**nodefs.inodesFree** = **node.stats.fs.inodesFree**	
	imagefs.available	**imagefs.available** = **node.stats.runtime.imagefs.available**	
	imagefs.inodesFree	**imagefs.inodesFree** = **node.stats.runtime.imagefs.inodesFree**	

上記の表のそれぞれのシグナルは、literal または percentage ベースの値のいずれかをサポートします (**inodesFree** は除く)。**inodesFree** シグナルはパーセンテージとして指定する必要があります。パーセンテージベースの値は、各シグナルに関連付けられる合計容量との関連で計算されます。

スクリプトは kubelet が実行する一連の手順を使用し、cgroup から **memory.available** 値を派生させます。スクリプトは計算から非アクティブなファイルメモリー (つまり、非アクティブな LRU リストのファイルベースのメモリーのバイト数) を計算から除外します。 非アクティブなファイルメモリーはリソースの不足時に回収可能になることが想定されます。

注記

free -m はコンテナーで機能しないため、**free -m** のようなツールは使用しないでください。

OpenShift Container Platform はこれらのファイルシステムを 10 秒ごとにモニターします。

ボリュームおよびログを専用ファイルシステムに保存する場合、ノードはそのファイルシステムをモニターしません。

注記

ノードは、ディスク不足に基づくエビクションの意思決定をトリガーする機能をサポートします。ディスク不足のために Pod をエビクトする前に、ノードは コンテナーおよびイメージのガベージコレクション も実行します。

24.2.3. エビクションのしきい値について

ノードを設定してエビクションしきい値を指定できます。しきい値に達すると、リソースを回収するようにノードがトリガーされます。ノード設定ファイル でしきい値を設定することができます。

関連付けられた猶予期間とは別にエビクションのしきい値に達する場合、ノードは、ノードがメモリー不足またはディスク不足であることを示す状態を報告します。不足を報告することで、リソースの回収が試行されている間、スケジューラーはノード上の追加の Pod をスケジュールすることができなくなります。

ノードは、**node-status-update-frequency** 引数で指定された頻度で、ノードのステータス更新を継続的に報告します。デフォルトの頻度は **10s** (10 秒) です。

エビクションのしきい値は、しきい値に達する際にノードが即時にアクションを実行する場合に ハード となり、リソース回収前の猶予期間を許可する場合は ソフト になります。

注記

ソフトエビクションの使用は、特定のレベルの使用率をターゲットにする場合により一般的ですが、一時的な値の上昇を許容できます。ソフトエビクションは、ハードエビクションのしきい値よりも低く設定することが推奨されますが、期間はオペレーターが固有に設定できます。システム予約もソフトエビクションのしきい値以上に設定する必要があります。

ソフトエビクションのしきい値は高度な機能になります。ソフトエビクションのしきい値の使用を試行する前にハードエビクションのしきい値を設定してください。

しきい値は以下の形式で設定されます。

```
<eviction_signal><operator><quantity>
```

- **eviction-signal** 値は任意の サポートされるエビクションシグナル にすることができます。
- **operator** 値は **<** になります。
- **quantity** 値は、Kubernetes で使用される 数量表現 と一致している必要があり、**%** トークンで終了する場合はパーセンテージで表現することができます。

たとえば、オペレーターが 10Gi メモリーのあるノードを持つ場合で、オペレーターは利用可能なメモリーが 1Gi を下回る場合にエビクションを導入する必要がある場合、メモリーのエビクションしきい値は以下のいずれかで指定することができます。

```
memory.available<1Gi
memory.available<10%
```

注記

ノードはエビクションしきい値の評価とモニターを 10 秒ごとに実行し、値を変更することはできません。これはハウスキープ処理の間隔になります。

24.2.3.1. ハードエビクションのしきい値について

ハードエビクションのしきい値には猶予期間がありません。ハードエビクションのしきい値に達すると、ノードは関連付けられたリソースを回収するために即時のアクションを実行します。たとえば、ノードは正常に終了せずに、1 つ以上の Pod をすぐに終了できます。

ハードエビクションのしきい値を設定するには、ポリシー作成のためのノード設定の使用 に示されるように、エビクションしきい値を **eviction-hard** の下にある ノード設定ファイル に追加します。

ハードエビクションのしきい値が設定されたノード設定ファイルのサンプル

```
kubeletArguments:
  eviction-hard:
  - memory.available<500Mi
  - nodefs.available<500Mi
  - nodefs.inodesFree<5%
  - imagefs.available<100Mi
  - imagefs.inodesFree<10%
```

この例は一般的なガイドラインを示すためのもので、推奨される設定ではありません。

24.2.3.1.1. デフォルトのハードエビクションしきい値

OpenShift Container Platform は、**eviction-hard** に以下のデフォルト設定を使用します。

```
...
kubeletArguments:
  eviction-hard:
  - memory.available<100Mi
  - nodefs.available<10%
  - nodefs.inodesFree<5%
  - imagefs.available<15%
...
```

24.2.3.2. ソフトエビクションのしきい値について

ソフトエビクションのしきい値は、エビクションしきい値と管理者が指定する必要な猶予期間のペアを設定します。ノードは、猶予期間が経過するまではエビクションシグナルに関連付けられたリソースを回収しません。ノード設定で猶予期間が指定されていない場合、ノードは起動時にエラーを生成します。

さらに、ソフトエビクションのしきい値に達する場合、Operator は Pod をノードからエビクトする際に使用する Pod 終了の猶予期間として許可される最長期間を指定できます。**eviction-max-pod-grace-period** が指定されると、ノードは **pod.Spec.TerminationGracePeriodSeconds** と maximum-allowed-grace-period の値の小さい方を使用します。これが指定されていない場合は、ノードは正常な停止なしに、Pod を即時に強制終了します。

ソフトエビクションのしきい値については、以下のフラグがサポートされています。

- **eviction-soft**: **memory.available<1.5Gi** などのエビクションしきい値のセット。しきい値が対応する猶予期間で満たされる場合、しきい値は Pod のエビクションをトリガーします。
- **eviction-soft-grace-period**: **memory.available=1m30s** などのエビクションの猶予期間のセット。猶予期間は、Pod のエビクションをトリガーするまでソフトエビクションのしきい値が保持される期間に対応します。
- **eviction-max-pod-grace-period**: ソフトエビクションのしきい値に達する際の Pod の終了時に使用される最長で許可される猶予期間 (秒単位) です。

ソフトエビクションのしきい値を設定するには、ポリシー作成のためのノード設定の使用 に示されるように、エビクションのしきい値を **eviction-soft** の下にある ノード設定ファイル に追加します。

ソフトエビクションのしきい値が設定されたノード設定ファイルのサンプル

```
kubeletArguments:
  eviction-soft:
  - memory.available<500Mi
  - nodefs.available<500Mi
  - nodefs.inodesFree<5%
  - imagefs.available<100Mi
  - imagefs.inodesFree<10%
  eviction-soft-grace-period:
  - memory.available=1m30s
  - nodefs.available=1m30s
  - nodefs.inodesFree=1m30s
  - imagefs.available=1m30s
  - imagefs.inodesFree=1m30s
```

この例は一般的なガイドラインを示すためのもので、推奨される設定ではありません。

24.3. スケジューリング用のリソース量の設定

スケジューラーがノードを完全に割り当て、エビクションを防止できるようにするために、スケジューリングで利用できるノードリソースの数量を制御できます。

system-reserved を、Pod のデプロイおよび system-daemon 用にスケジューラーで利用可能にするリソース量と等しくなるようにします。**system-reserved** リソースは、**sshd** および **NetworkManager** などのオペレーティングシステムのデーモン用に予約されています。エビクションは、Pod が割り当て

可能なリソースの要求量よりも多くのリソースを使用する場合に生じます。

ノードは 2 つの値を報告します。

- **Capacity**: マシンにあるリソースの量です。
- **Allocatable**: スケジューリング用に利用できるリソースの量です。

割り当て可能なリソースの量を設定するには、適切な ノード設定マップ を編集して、**eviction-hard** または **eviction-soft** の **system-reserved** パラメーターを追加するか、または変更します。

```
kubeletArguments:
  eviction-hard: 1
    - "memory.available<500Mi"
  system-reserved:
    - "memory=1.5Gi"
```

1 このしきい値は、**eviction-hard** または **eviction-soft** のいずれかになります。

system-reserved 設定に適切な値を決定するには、ノード要約 API を使用してノードのリソース使用状況を判別します。詳細は、割り当てられたリソース用のノードの設定 を参照してください。

変更を有効にするために OpenShift Container Platform サービスを再起動します。

```
# systemctl restart atomic-openshift-node
```

24.4. ノードの状態変動の制御

ノードがソフトエビクションしきい値の上下で変動している場合に、関連する猶予期間を超えていない場合、この変動により、スケジューラーの問題が生じる可能性があります。

この変動を防ぐには、**eviction-pressure-transition-period** パラメーターを設定して、ノードが不足状態からの切り換え前に待機する期間を制御します。

1. **<resource_type>=<resource_quantity>** ペアのセットを使用して、適切な ノード設定マップ の **kubeletArguments** セクションにパラメーターを編集または追加します。

   ```
   kubeletArguments:
     eviction-pressure-transition-period:
     - 5m
   ```

   ノードは、指定期間の指定された不足状態についてエビクションしきい値に達していない場合は状態を false に戻します。

   注記

   調整を行う前にデフォルト値 (5 分) を使用します。デフォルト値は、システムを安定させ、スケジューラーがノードが安定する前にノードに新しい Pod をスケジュールするのを防ぐことを目的としています。

2. 変更を有効するために OpenShift Container Platform サービスを再起動します。

   ```
   # systemctl restart atomic-openshift-node
   ```

24.5. ノードレベルのリソースの回収

エビクション条件が満たされる場合、ノードはシグナルが定義されたしきい値を下回るまで、不足状態にあるリソースを回収するプロセスを実行します。この間、ノードはいずれの新規 Pod のスケジューリングもサポートしません。

ノードは、ホストシステムにコンテナーランタイム用の専用の **imagefs** が設定されているかどうかに基づいて、ノードがエンドユーザー Pod をエビクトする前にノードレベルのリソースを回収しようとします。

Imagefs が設定されている場合
ホストシステムに **imagefs** が設定されている場合:

- **nodefs** ファイルシステムがエビクションしきい値を満たす場合、ノードは以下の順番でディスク領域を解放します。
    - 実行されない Pod およびコンテナーを削除します。
- **imagefs** ファイルシステムがエビクションのしきい値を満たす場合、ノードは以下の順番でディスク領域を解放します。
    - すべての未使用イメージを削除します。

Imagefs が設定されていない場合
ホストシステムに **imagefs** がない場合:

- **nodefs** ファイルシステムがエビクションしきい値を満たす場合、ノードは以下の順番でディスク領域を解放します。
    - 実行されない Pod およびコンテナーを削除します。
    - すべての未使用イメージを削除します。

24.6. POD エビクションについて

エビクションしきい値に達し、猶予期間を経過している場合、ノードはシグナルが定義されたしきい値を下回るまで Pod のエビクトのプロセスを実行します。

ノードは、エビクション用に Pod を QoS (Quality of Service) でランク付けします。同じ QoS (Quality of Service) を持つ Pod において、ノードは Pod のスケジューリング要求に対するコンピュートリソースの消費によって Pod をランク付けします。

各 QoS にはメモリー不足のスコアがあります。Linux out-of-memory ツール (OOM killer) はスコアを使用して、終了する Pod を判別します。詳細は、QoS および Out of Memory Killer について を参照してください。

次の表に、各 QoS (Quality of Service) レベルと関連するメモリー不足スコアを示します。

表24.2 Quality of Service (QoS) レベル

QoS (Quality of Service)	説明
Guaranteed	要求に対してリソースを最も多く消費する Pod が最初に失敗します。いずれの Pod も要求を超えていない場合、ストラテジーはリソースの最大コンシューマーをターゲットにします。

QoS (Quality of Service)	説明
Burstable	そのリソースの要求に対してリソースを最も多く消費する Pod が最初に失敗します。いずれの Pod も要求を超えていない場合、ストラテジーはリソースの最大コンシューマーをターゲットにします。
BestEffort	リソースの最大量を消費する Pod が最初に失敗します。

ノードやコンテナーエンジンなどのシステムデーモンが **system-reserved** 割り当てを使用して、予約されたよりも多くのリソースを消費しない場合や、ノードに Guaranteed QoS (Quality of Service) Pod のみが残っている場合は、Guaranteed QoS (Quality of Service) Pod が別の Pod によるリソース消費のためにエビクトされることはありません。

ノードに Guaranteed QoS (Quality of Service) Pod のみが残っている場合、ノードはノードの安定性に最も影響の少ない Pod をエビクトし、他の Guaranteed QoS (Quality of Service) Pod に対する予想外の消費による影響を制限します。

ローカルディスクは、best-effort の QoS (Quality of Service) リソースです。必要な場合は、ノードはディスク不足の発生時にディスク領域を回収するために Pod を 1 度に 1 つずつエビクトします。ノードは QoS に基づいて Pod をランク付けします。ノードが空き inode の不足に対応している場合、ノードは QoS (Quality of Service) が最も低い Pod を最初にエビクトして inode を回収します。ノードが利用可能なディスクの不足状態に対応している場合、ノードはローカルディスクを最も多く消費する QoS 内の Pod をランク付けし、それらの Pod を先にエビクトします。

24.6.1. QoS および Out of Memory Killer について

ノードによるメモリーの回収が可能になる前に、システムの OOM (Out of Memory) イベントが発生する場合、ノードは OOM killer に依存して応答します。

ノードは、Pod の QoS に基づいて各コンテナーの **oom_score_adj** 値を設定します。

表24.3 Quality of Service (QoS) レベル

QoS (Quality of Service)	**oom_score_adj** Value
Guaranteed	-998
Burstable	min(max(2, 1000 - (1000 * memoryRequestBytes) / machineMemoryCapacityBytes), 999)
BestEffort	1000

ノードがシステムの OOM イベントに直面する前にノードがメモリーを回収できない場合、OOM killer プロセスは OOM スコアを計算します。

```
% of node memory a container is using + oom_score_adj = oom_score
```

次にノードは、スコアの最も高いコンテナーを終了します。

スケジューリング要求に関連してメモリーを最も多く消費する、QoS (Quality of Service) の最も低いレベルにあるコンテナーが最初に終了します。

Pod の削除とは異なり、Pod コンテナーが OOM が原因で終了する場合、ノードはノードの再起動ポリシーに従ってコンテナーを再起動できます。

24.7. POD スケジューラーおよび OOR 状態について

スケジューラーは、スケジューラーが追加の Pod をノードに配置する際にノードの状態を表示します。たとえば、ノードに以下のようなエビクションのしきい値がある場合は、以下のようになります。

```
eviction-hard is "memory.available<500Mi"
```

さらに利用可能なメモリーが 500Mi を下回る場合、ノードは **Node.Status.Conditions** の **MemoryPressure** の値を true として報告します。

表24.4 ノードの状態およびスケジューラーの動作

ノードの状態	スケジューラーの動作
MemoryPressure	ノードがこの状態を報告する場合、スケジューラーは **BestEffort** Pod をそのノードに配置しません。
DiskPressure	ノードがこの状態を報告する場合、スケジューラーは追加の Pod をノードに配置しません。

24.8. シナリオ例

Operator:

- メモリー容量が 10Gi のノードがある。
- システムデーモン (カーネル、ノード、他のデーモンなど) のメモリー容量の 10% を予約する必要がある。
- システムの OOM の悪化または発生の可能性を軽減するため、メモリー使用率が 95% の時点で Pod をエビクトする必要がある。

この設定から、**system-reserved** にはエビクションのしきい値でカバーされるメモリー量が含まれていることを読み取ることができます。

この容量に達するのは、一部の Pod による使用がその要求を超えるか、またはシステムによる使用が 1Gi を超える場合のいずれかになります。

ノードに 10 Gi の容量があり、システムデーモン (**system-reserved** が設定されている) に 10% の容量を予約する必要がある場合、以下の計算を実行します。

```
capacity = 10 Gi
system-reserved = 10 Gi * .1 = 1 Gi
```

割り当て可能なリソースの量は以下のようになります。

```
allocatable = capacity - system-reserved = 9 Gi
```

これは、デフォルトでスケジューラーはノードに対し、9 Gi のメモリーを要求する Pod をスケジュールすることを意味します。

使用可能なメモリーが 30 秒間で容量の 10% を下回ることをノードが検出する際や、容量の 5% を下回る際にすぐにエビクションがトリガーされるようにエビクションを有効にする必要がある場合は、スケジューラーで 8Gi が割り当て可能であることを確認する必要があります。そのため、システム予約ではエビクションしきい値の大きい方の値がカバーされている必要があります。

```
capacity = 10 Gi
eviction-threshold = 10 Gi * .1 = 1 Gi
system-reserved = (10Gi * .1) + eviction-threshold = 2 Gi
allocatable = capacity - system-reserved = 8 Gi
```

以下を適切な ノード設定マップ に追加します。

```
kubeletArguments:
  system-reserved:
  - "memory=2Gi"
  eviction-hard:
  - "memory.available<.5Gi"
  eviction-soft:
  - "memory.available<1Gi"
  eviction-soft-grace-period:
  - "memory.available=30s"
```

この設定により、スケジューラーは Pod をノードに配置せず、メモリー不足をすぐに発生させ、エビクションをトリガーさせます。この設定は、これらの Pod が設定された要求よりも少ない量を使用することを前提としています。

24.9. 推奨される対策

24.9.1. DaemonSet および Out of Resource (リソース不足) の処理

ノードがデーモンセットによって作成された Pod をエビクトすると、Pod は即時に再作成され、同じノードに再スケジュールされます。スケジューラーは、デーモンセットによって作成される Pod と他のオブジェクトを区別できないため、このように動作します。

通常、デーモンセットは、作成する Pod がエビクションの候補として特定されることを防ぐため、best effort Pod を作成するべきではありません。代わりに、デーモンセットは Pod を起動し、それらを 保証された QoS (Quality of Service) で設定する必要があります。

第25章 制限範囲の設定

25.1. 制限範囲の目的

LimitRange オブジェクトで定義される制限範囲は、Pod、コンテナー、イメージ、イメージストリーム、および Persistent Volume Claim (永続ボリューム要求、PVC) のレベルで プロジェクト の コンピュートリソース制約 を列挙し、Pod、コンテナー、イメージ、イメージストリームまたは Persistent Volume Claim (永続ボリューム要求、PVC) で消費できるリソースの量を指定します。

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。リソースが明示的な値を指定しない場合で、制約がデフォルト値をサポートする場合は、デフォルト値がリソースに適用されます。

CPU とメモリーの制限について、最大値を指定しても最小値を指定しない場合、リソースは最大値よりも多くの CPU とメモリーリソースを消費する可能性があります。

一時ストレージのテクノロジープレビューを使用して一時ストレージの制限と要求を指定できます。この機能はデフォルトでは無効になっています。この機能を有効にするには、configuring for ephemeral storage を参照してください。

コア Limit Range オブジェクトの定義

```
apiVersion: "v1"
kind: "LimitRange"
metadata:
  name: "core-resource-limits" (1)
spec:
  limits:
    - type: "Pod"
      max:
        cpu: "2" (2)
        memory: "1Gi" (3)
      min:
        cpu: "200m" (4)
        memory: "6Mi" (5)
    - type: "Container"
      max:
        cpu: "2" (6)
        memory: "1Gi" (7)
      min:
        cpu: "100m" (8)
        memory: "4Mi" (9)
      default:
        cpu: "300m" (10)
        memory: "200Mi" (11)
      defaultRequest:
        cpu: "200m" (12)
        memory: "100Mi" (13)
      maxLimitRequestRatio:
        cpu: "10" (14)
```

1. 制限範囲オブジェクトの名前です。
2. すべてのコンテナーにおいて Pod がノードで要求できる CPU の最大量です。
3. すべてのコンテナーにおいて Pod がノードで要求できるメモリーの最大量です。
4. すべてのコンテナーにおいて Pod がノードで要求できる CPU の最小量です。**min** 値を設定しない場合や、**min** を **0** に設定すると、結果は制限されず、Pod は **max** CPU 値を超える量を消費することができます。
5. すべてのコンテナーにおいて Pod がノードで要求できるメモリーの最小量です。**min** 値を設定しない場合や、**min** を **0** に設定すると、結果は制限されず、Pod は **max** メモリー値を超える量を消費することができます。
6. Pod の単一コンテナーが要求できる CPU の最大量です。
7. Pod の単一コンテナーが要求できるメモリーの最大量です。
8. Pod の単一コンテナーが要求できる CPU の最小量です。**min** 値を設定しない場合や、**min** を **0** に設定すると、結果は制限されず、Pod は **max** CPU 値を超える量を消費することができます。
9. Pod の単一コンテナーが要求できるメモリーの最小量です。**min** 値を設定しない場合や、**min** を **0** に設定すると、結果は制限されず、Pod は **max** メモリー値を超える量を消費することができます。
10. Pod 仕様で制限を指定しない場合の、コンテナーのデフォルトの CPU 制限。
11. Pod 仕様で制限を指定しない場合の、コンテナーのデフォルトのメモリー制限。
12. Pod 仕様で要求を指定しない場合の、コンテナーのデフォルトの CPU 要求。
13. Pod 仕様で要求を指定しない場合の、コンテナーのデフォルトのメモリー要求。
14. コンテナーの要求に対する制限の最大比率。

CPU およびメモリーの測定方法についての詳細は、コンピュートリソース を参照してください。

OpenShift Container Platform の Limit Range オブジェクトの定義

```
apiVersion: "v1"
kind: "LimitRange"
metadata:
  name: "openshift-resource-limits"
spec:
  limits:
    - type: openshift.io/Image
      max:
        storage: 1Gi (1)
    - type: openshift.io/ImageStream
      max:
        openshift.io/image-tags: 20 (2)
        openshift.io/images: 30 (3)
    - type: "Pod"
      max:
        cpu: "2" (4)
```

```
      memory: "1Gi" 5
      ephemeral-storage: "1Gi" 6
    max:
      cpu: "1" 7
      memory: "1Gi" 8
```

1. 内部レジストリーにプッシュできるイメージの最大サイズ。
2. イメージストリームの仕様で定義される一意のイメージタグの最大数。
3. イメージストリームのステータスについて仕様で定義される一意のイメージ参照の最大数。
4. すべてのコンテナーにおいて Pod がノードで要求できる CPU の最大量です。
5. すべてのコンテナーにおいて Pod がノードで要求できるメモリーの最大量です。
6. 一時ストレージのテクノロジープレビュー機能が有効にされる場合に、すべてのコンテナーにおいて Pod がノードで要求できる一時ストレージの最大量です。
7. すべてのコンテナーにおいて Pod がノードで要求できる CPU の最小量です。**min** 値を設定する場合や、**min** を **0** に設定すると、結果は制限されず、Pod は **max** CPU 値を超える量を消費することができます。
8. すべてのコンテナーにおいて Pod がノードで要求できるメモリーの最小量です。**min** 値を設定しない場合や、**min** を **0** に設定すると、結果の制限がなく、Pod は **max** メモリー値を超える量を消費することができます。

コアおよび OpenShift Container Platform リソースの両方を 1 つの制限範囲オブジェクトで指定できます。これらは、明確にするために 2 つの例に個別に示します。

25.1.1. コンテナーの制限

サポートされるリソース:

- CPU
- メモリー

サポートされる制約:

コンテナーごとに設定されます。 指定される場合、以下を満たしている必要があります。

表25.1 コンテナー

制約	動作
Min	**Min[resource]**: **container.resources.requests[resource]** (必須) または **container/resources.limits[resource]** (オプション) 以下 設定で **min** CPU を定義している場合、要求値はその CPU 値よりも大きくなければなりません。**min** 値を設定しない場合や、**min** を **0** に設定すると、結果は制限されず、Pod は **max** 値よりも多くのリソースを消費できます。

制約	動作
Max	**container.resources.limits[resource]** (必須): **Max[resource]** 以下 設定で **max** CPU を定義している場合、CPU 要求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 制約を満たす制限を設定する必要があります。
MaxLimitRequestRatio	**MaxLimitRequestRatio[resource]** は (**container.resources.limits[resource]** / **container.resources.requests[resource]**) 以下です。 制限範囲で **maxLimitRequestRatio** 制約を定義する場合、新規コンテナーには **request** と **limit** 値の両方が必要になります。さらに OpenShift Container Platform 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。結果は、1 を超える整数である必要があります。 たとえば、コンテナーの **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。

サポートされるデフォルト:

Default[resource]
指定がない場合は **container.resources.limit[resource]** を所定の値にデフォルト設定します。

Default Requests[resource]
指定がない場合は、**container.resources.requests[resource]** を所定の値にデフォルト設定します。

25.1.2. Pod の制限

サポートされるリソース:

- CPU
- メモリー

サポートされる制約:

Pod のすべてのコンテナーにおいて、以下を満たしている必要があります。

表25.2 Pod

制約	実施される動作
Min	**Min[resource]**: **container.resources.requests[resource]** (必須) 以下または **container.resources.limits[resource]** 以下**min** 値を設定しない場合や、**min** を **0** に設定すると、結果は制限されず、Pod は **max** 値よりも多くのリソースを消費できます。

制約	実施される動作
Max	**container.resources.limits[resource]** (必須): **Max[resource]** 以下
MaxLimitRequestRatio	**MaxLimitRequestRatio[resource]** は (**container.resources.limits[resource]** / **container.resources.requests[resource]**) 以下です。

25.1.3. イメージの制限

サポートされるリソース:

- ストレージ

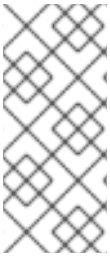
リソースタイプ名:

- **openshift.io/Image**

イメージごとに設定されます。 指定される場合、以下が一致している必要があります。

表25.3 イメージ

制約	動作
Max	**image.dockerimagemetadata.size**: **Max[resource]** より小さいか等しい



注記

制限を超える Blob がレジストリーにアップロードされないようにするために、クォータを実施するようレジストリーを設定する必要があります。**REGISTRY_MIDDLEWARE_REPOSITORY_OPENSHIFT_ENFORCEQUOTA** 環境変数を **true** に設定する必要があります。デフォルトでは、新規デプロイメントでは、環境変数は **true** に設定されます。

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、すべてのサイズ情報は含まれません。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。

25.1.4. イメージストリームの制限

サポートされるリソース:

- **openshift.io/image-tags**
- **openshift.io/images**

リソースタイプ名:

- **openshift.io/ImageStream**

イメージストリームごとに設定されます。 指定される場合、以下が一致している必要があります。

表25.4 ImageStream

制約	動作
Max[openshift.io/image-tags]	**length(uniqueimagetags(imagestream.spec.tags))**: **Max[openshift.io/image-tags]** より小さいか等しい **uniqueimagetags** は、指定された仕様タグのイメージへの一意の参照を返します。
Max[openshift.io/images]	**length(uniqueimages(imagestream.status.tags))**: **Max[openshift.io/images]** より小さいか等しい **uniqueimages** はステータスタグにある一意のイメージ名を返します。名前はイメージのダイジェストと等しくなります。

25.1.4.1. イメージ参照の数

openshift.io/image-tags リソースは、固有の イメージ参照 を表します。使用可能な参照は、**ImageStreamTag**、**ImageStreamImage**、または **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、または タグのトラッキング 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、イメージストリーム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。内部コンテナー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、内部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。

25.1.5. PersistentVolumeClaim の制限

サポートされるリソース:

- ストレージ

サポートされる制約:

プロジェクトのすべての Persistent Volume Claim (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。

表25.5 Pod

制約	実施される動作
Min	Min[resource] <= claim.spec.resources.requests[resource] (必須)
Max	claim.spec.resources.requests[resource] (必須) <= Max[resource]

Limit Range オブジェクトの定義

```
{
  "apiVersion": "v1",
  "kind": "LimitRange",
  "metadata": {
    "name": "pvcs" 1
  },
  "spec": {
    "limits": [{
        "type": "PersistentVolumeClaim",
        "min": {
          "storage": "2Gi" 2
        },
        "max": {
          "storage": "50Gi" 3
        }
      }
    ]
  }
}
```

1. 制限範囲オブジェクトの名前です。
2. Persistent Volume Claim (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。

25.2. 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. 必要な仕様で制限範囲オブジェクト定義を作成します。
2. オブジェクトを作成します。

```
$ oc create -f <limit_range_file> -n <project>
```

25.3. 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範囲を表示できます。

以下の手順を実行して、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制限範囲オブジェクトの一覧を取得します。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
```

出力例

```
NAME              AGE
resource-limits   6d
```

2. 制限範囲を記述します。たとえば、**resource-limits** という制限範囲の場合:

```
$ oc describe limits resource-limits -n demoproject
```

出力例

```
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      -               -               -
```

25.4. 制限範囲の削除

制限範囲を削除し、プロジェクトの制限を実行しないようにするには、以下を実行します。

- 次のコマンドを実行します。

```
$ oc delete limits <limit_name>
```

第26章 NODE PROBLEM DETECTOR

26.1. 概要

Node Problem Detector (ノード問題検出機能) は特定の問題を検出し、それらの問題を API サーバーに報告することで、ノードの正常性をモニターします。Node Problem Detector は、各ノードで daemonSet として実行されます。

重要

Node Problem Detector 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 の実稼働環境でのサービスレベルアグリーメント (SLA) ではサポートされていないため、Red Hat では実稼働環境での使用を推奨していません。テクノロジープレビューの機能は、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についての詳細は、https://access.redhat.com/support/offerings/techpreview/ を参照してください。

Node Problem Detector はシステムログを読み取り、特定のエントリーの有無を監視し、コントロールプレーンにそれらの問題を表示します。これは、**oc get node** および **oc get event** などの OpenShift Container Platform のコマンドを使用して表示することができます。これらの問題については、適宜修正するようアクションを実行するか、または OpenShift Container Platform ログモニターリング などの選択可能なツールを使用して、メッセージをキャプチャーすることができます。検出される問題は以下のいずれかのカテゴリーに分類できます。

- **NodeCondition**: ノードを Pod に対して利用不可にする永続的な問題です。ノードの状態は、ホストが再起動されるまでクリアされません。
- **Event**: ノードに制限的な影響を与える一時的な問題で、情報を提供します。

Node Problem Detector は以下を検出できます。

- コンテナーランタイムの問題:
  - 反応しないランタイムデーモン
- ハードウェアの問題:
  - 正常でない CPU
  - 正常でないメモリー
  - 正常でないディスク
- カーネルの問題:
  - カーネルのデッドロック状態
  - 破損したファイルシステム
  - 反応しないランタイムデーモン
- インフラストラクチャーデーモンの問題:
  - NTP サービスの停止

26.2. NODE PROBLEM DETECTOR の出力サンプル

以下の例では、特定のノードでカーネルのデッドロックを監視する Node Problem Detector の出力を示しています。コマンドでは **oc get node** を使用し、ログで **KernelDeadlock** エントリーについてフィルターし、特定のノードを監視します。

```
# oc get node <node> -o yaml | grep -B5 KernelDeadlock
```

Node Problem Detector の出力サンプル (問題がない場合)

```
message: kernel has no deadlock
reason: KernelHasNoDeadlock
status: false
type: KernelDeadLock
```

KernelDeadLock 状態の出力サンプル

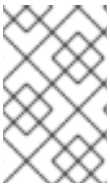
```
message: task docker:1234 blocked for more than 120 seconds
reason: DockerHung
status: true
type: KernelDeadLock
```

この例は、ノードでイベントの有無を監視する Node Problem Detector からの出力を示しています。以下のコマンドでは、デフォルト プロジェクトに対して **oc get event** を使用し、Node Problem Detector 設定マップ の **kernel-monitor.json** セクションに一覧表示されているイベントの有無を監視します。

```
# oc get event -n default --field-selector=source=kernel-monitor --watch
```

ノードのイベントを表示する出力サンプル

```
LAST SEEN                   FIRST SEEN                  COUNT NAME     KIND  SUBOBJECT TYPE
REASON      SOURCE                  MESSAGE
2018-06-27 09:08:27 -0400 EDT   2018-06-27 09:08:27 -0400 EDT 1     my-node1 node
Warning TaskHunk    kernel-monitor.my-node1  docker:1234 blocked for more than 300 seconds
2018-06-27 09:08:27 -0400 EDT   2018-06-27 09:08:27 -0400 EDT 3     my-node2 node
Warning KernelOops  kernel-monitor.my-node2  BUG: unable to handle kernel NULL pointer
deference at nowhere
2018-06-27 09:08:27 -0400 EDT   2018-06-27 09:08:27 -0400 EDT 1     my-node1 node
Warning KernelOops  kernel-monitor.my-node2  divide error 0000 [#0] SMP
```

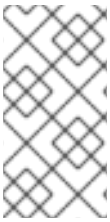


注記

Node Problem Detector はリソースを消費します。Node Problem Detector を使用する場合は、クラスターパフォーマンスのバランスを取るのに十分なノードがあることを確認します。

26.3. NODE PROBLEM DETECTOR のインストール

openshift_node_problem_detector_install が **/etc/ansible/hosts** インベントリーファイルで **true** に設定されていた場合、インストール では、デフォルトで Node Problem Detector の Deamonset を作成し、**openshift-node-problem-detector** という Detector のプロジェクトを作成します。



注記

Node Problem Detector はテクノロジープレビューとして提供されているため、**openshift_node_problem_detector_install** はデフォルトで **false** に設定されています。Node Problem Detector のインストール時には、パラメーターを **true** に手動で設定する必要があります。

Node Problem Detector がインストールされていない 場合、Playbook ディレクトリーに切り替え、**openshift-node-problem-detector/config.yml** Playbook を実行して Node Problem Detector をインストールします。

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook playbooks/openshift-node-problem-detector/config.yml
```

26.4. 検出された条件のカスタマイズ

Node Problem Detector 設定マップを編集し、Node Problem Detector をログの文字列を監視するように設定できます。

Node Problem Detector 設定マップのサンプル

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: node-problem-detector
data:
  docker-monitor.json: | (1)
    {
        "plugin": "journald", (2)
        "pluginConfig": {
                "source": "docker"
        },
        "logPath": "/host/log/journal", (3)
        "lookback": "5m",
        "bufferSize": 10,
        "source": "docker-monitor",
        "conditions": [],
        "rules": [ (4)
                {
                        "type": "temporary", (5)
                        "reason": "CorruptDockerImage", (6)
                        "pattern": "Error trying v2 registry: failed to register layer: rename
/var/lib/docker/image/(.+) /var/lib/docker/image/(.+): directory not empty.*" (7)
                }
        ]
    }
  kernel-monitor.json: | (8)
    {
        "plugin": "journald", (9)
        "pluginConfig": {
                "source": "kernel"
        },
        "logPath": "/host/log/journal", (10)
```

```
        "lookback": "5m",
        "bufferSize": 10,
        "source": "kernel-monitor",
        "conditions": [                    11
                {
                        "type": "KernelDeadlock", 12
                        "reason": "KernelHasNoDeadlock", 13
                        "message": "kernel has no deadlock"  14
                }
        ],
        "rules": [
                {
                        "type": "temporary",
                        "reason": "OOMKilling",
                        "pattern": "Kill process \\d+ (.+) score \\d+ or sacrifice child\\nKilled process \\d+ (.+)
total-vm:\\d+kB, anon-rss:\\d+kB, file-rss:\\d+kB"
                },
                {
                        "type": "temporary",
                        "reason": "TaskHung",
                        "pattern": "task \\S+:\\w+ blocked for more than \\w+ seconds\\."
                },
                {
                        "type": "temporary",
                        "reason": "UnregisterNetDevice",
                        "pattern": "unregister_netdevice: waiting for \\w+ to become free. Usage count = \\d+"
                },
                {
                        "type": "temporary",
                        "reason": "KernelOops",
                        "pattern": "BUG: unable to handle kernel NULL pointer dereference at .*"
                },
                {
                        "type": "temporary",
                        "reason": "KernelOops",
                        "pattern": "divide error: 0000 \\[#\\d+\\] SMP"
                },
                {
                        "type": "permanent",
                        "condition": "KernelDeadlock",
                        "reason": "AUFSUmountHung",
                        "pattern": "task umount\\.aufs:\\w+ blocked for more than \\w+ seconds\\."
                },
                {
                        "type": "permanent",
                        "condition": "KernelDeadlock",
                        "reason": "DockerHung",
                        "pattern": "task docker:\\w+ blocked for more than \\w+ seconds\\."
                }
        ]
}
```

1 コンテナーイメージに適用されるルールおよび条件。

2 9 コンマ区切りの一覧のモニターリングサービス

3 10 モニターリングサービスログのパス。

4 11 モニターされるイベントの一覧。

5 12 エラーがイベント (**temporary**) または NodeCondition (**permanent**) であることを示すラベル。

6 13 エラーを記述するテキストメッセージ。

7 14 Node Problem Detector が監視するエラーメッセージ。

8 カーネルに適用されるルールおよび条件。

Node Problem Detector を設定するには、問題の状態およびイベントを追加するか、またはこれらを削除します。

1. テキストエディターで Node Problem Detector 設定マップを編集します。

```
$ oc edit configmap -n openshift-node-problem-detector node-problem-detector
```

2. ノードの状態またはイベントを必要に応じて削除、追加、または編集します。

```
{
      "type": <`temporary` or `permanent`>,
      "reason": <free-form text describing the error>,
      "pattern": <log message to watch for>
},
```

以下に例を示します。

```
{
      "type": "temporary",
      "reason": "UnregisterNetDevice",
      "pattern": "unregister_netdevice: waiting for \\w+ to become free. Usage count = \\d+"
},
```

3. 変更を適用するために実行中の Pod を再起動します。Pod を再起動するために、すべての既存 Pod を削除できます。

```
# oc delete pods -n openshift-node-problem-detector -l name=node-problem-detector
```

4. Node Problem Detector 出力を標準出力 (stdout) および標準エラー出力 (stderr) に表示するには、以下を Node Problem Detector の DaemonSet に追加します。

```
spec:
  template:
    spec:
      containers:
      - name: node-problem-detector
        command:
        - node-problem-detector
        - --alsologtostderr=true 1
        - --log_dir="/tmp" 2
        - --system-log-monitors=/etc/npd/kernel-monitor.json,/etc/npd/docker-monitor.json 3
```

1. 出力を標準出力 (stdout) に送信します。
2. エラーログへのパス。
3. プラグイン設定ファイルへのコンマ区切りのパス。

26.5. NODE PROBLEM DETECTOR が実行中であることの確認

Node Problem Detector が有効であることを確認するには、以下を実行します。

- 以下のコマンドを実行し、Problem Node Detector Pod の名前を取得します。

```
# oc get pods -n openshift-node-problem-detector

NAME                          READY     STATUS    RESTARTS   AGE
node-problem-detector-8z8r8   1/1       Running   0          1h
node-problem-detector-nggjv   1/1       Running   0          1h
```

- 以下のコマンドを実行し、Problem Node Detector Pod のログ情報を表示します。

```
# oc logs -n openshift-node-problem-detector <pod_name>
```

出力は以下のようになります。

```
# oc logs -n openshift-node-problem-detector node-problem-detector-c6kng
I0416 23:22:00.641354       1 log_monitor.go:63] Finish parsing log monitor config file:
{WatcherConfig:{Plugin:journald PluginConfig:map[source:kernel] LogPath:/host/log/journal
Lookback:5m} BufferSize:10 Source:kernel-monitor DefaultConditions:
[{Type:KernelDeadlock Status:false Transition:0001-01-01 00:00:00 +0000 UTC
Reason:KernelHasNoDeadlock Message:kernel has no deadlock}]
```

- ノードのイベントをシミュレーションして Node Problem Detector をテストします。

```
# echo "kernel: divide error: 0000 [#0] SMP." >> /dev/kmsg
```

- ノードの状態をシミュレーションして Node Problem Detector をテストします。

```
# echo "kernel: task docker:7 blocked for more than 300 seconds." >> /dev/kmsg
```

26.6. NODE PROBLEM DETECTOR のアンインストール

Node Problem Detector をアンインストールするには、以下を実行します。

1. Ansible インベントリーファイルに以下のオプションを追加します。

```
[OSEv3:vars]
openshift_node_problem_detector_state=absent
```

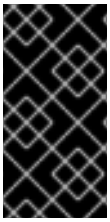
2. Playbook ディレクトリーに切り替え、**config.yml** Ansible Playbook を実行します。

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook playbooks/openshift-node-problem-detector/config.yml
```

第27章 INGRESS トラフィックの固有の外部 IP の割り当て

27.1. 概要

外部トラフィックをクラスターにつなぐ方法の 1 つとして、ExternalIP または IngressIP アドレスを使用することができます。

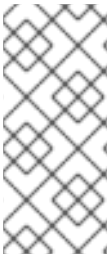


重要

この機能は、クラウド以外のデプロイメントでのみサポートされます。クラウド (GCE、AWS、および OpenStack) デプロイメントの場合、ロードバランサーサービスを使用し、クラウドの自動デプロイメントでサービスのエンドポイントをターゲットに設定します。

OpenShift Container Platform は 2 つの IP アドレスのプールをサポートします。

- IngressIP は、サービスの外部 IP アドレスを選択する場合に Loadbalancer で使用されます。
- ExternalIP は、ユーザーが設定されたプールから特定 IP を選択する場合に使用されます。



注記

これらはいずれも、ネットワークインターフェイスコントローラー (NIC) または仮想イーサネット、または外部ルーティングのいずれであっても、使用される OpenShift Container Platform ホストのデバイスに設定される必要があります。この場合、Ipfailover はホストを設定し、NIC を設定するため、これを使用することが推奨されます。

IngressIP および ExternalIP はいずれも外部トラフィックのクラスターへのアクセスを可能にし、適切にルーティングされている場合に、外部トラフィックはサービスが公開する TCP/UDP ポート経由でサービスのエンドポイントに到達できます。これは、外部 IP をサービスに手動で割り当てる際に、制限された数の共有 IP アドレスのポート領域を管理しなくてはならない場合よりも単純になります。またこれらのアドレスは、高可用性 を設定する場合に、仮想 IP (VIP) としても使用できます。

OpenShift Container Platform は IP アドレスの自動および手動割り当ての両方をサポートしており、それぞれのアドレスは 1 つのサービスの最大数に割り当てられることが保証されます。これにより、各サービスは、ポートが他のサービスで公開されているかによらず、自らの選択したポートを公開できます。

27.2. 制限

ExternalIP を使用するには、以下を実行できます。

- **externalIPNetworkCIDRs** 範囲から IP アドレスを選択します。
- マスター設定ファイルで、IP アドレスを **ingressIPNetworkCIDR** プールから割り当てます。この場合、OpenShift Container Platform はロードバランサーサービスタイプのクラウド以外のバージョンを実装し、IP アドレスをサービスに割り当てます。

  注意

  割り当てた IP アドレスがクラスター内の 1 つ以上のノードで終了することを確認する必要があります。既存の **oc adm ipfailover** を使用して、外部 IP の可用性が高いことを確認します。

手動で設定された外部 IP の場合、起こり得るポートのクラッシュについては first-come, first-served (先着順) で処理されます。ポートを要求する場合、その IP アドレスに割り当てられていない場合にのみ利用可能となります。以下に例を示します。

手動で設定された外部 IP のポートのクラッシュ例

2 つのサービスが同じ外部 IP アドレス 172.7.7.7 で手動で設定されている。

MongoDB service A がポート 27017 を要求し、次に **MongoDB service B** が同じポートを要求する。最初の要求がこのポートを取得します。

ただし、Ingress コントローラーが外部 IP を割り当てる場合、ポートのクラッシュは問題とはなりません。 コントローラーが各サービスに固有のアドレスを割り当てるためです。

27.3. 固有の外部 IP を使用するようクラスターを設定する

クラウド以外のクラスターで、**ingressIPNetworkCIDR** はデフォルトで **172.29.0.0/16** に設定されます。クラスター環境がこのプライベート範囲をまだ使用していない場合は、デフォルトを使用できます。ただし、異なる範囲を使用する必要がある場合は、ingress IP を割り当てる前に、**/etc/origin/master/master-config.yaml** ファイルで **ingressIPNetworkCIDR** を設定する必要があります。次に、マスターサービスを再起動します。

注意

LoadBalancer タイプのサービスに割り当てられる外部 IP は常に **ingressIPNetworkCIDR** の範囲にあります。**ingressIPNetworkCIDR** が割り当てられた外部 IP がこの範囲内からなくなるように変更される場合、影響を受けるサービスには、新規の範囲と互換性のある新規の外部 IP が割り当てられます。

注記

高可用性 を使用している場合、この範囲は 255 IP アドレス未満である必要があります。

/etc/origin/master/master-config.yaml のサンプル

```
networkConfig:
  ingressIPNetworkCIDR: 172.29.0.0/16
```

27.3.1. サービスの Ingress IP の設定

Ingress IP を割り当てるには、以下を実行します。

1. **loadBalancerIP** 設定で特定の IP を要求する LoadBalancer サービスの YAML ファイルを作成します。

   LoadBalancer 設定サンプル

   ```
   apiVersion: v1
   kind: Service
   metadata:
     name: egress-1
   spec:
     ports:
     - name: db
       port: 3306
   ```

```
  loadBalancerIP: 172.29.0.1
  type: LoadBalancer
  selector:
    name: my-db-selector
```

2. Pod に LoadBalancer サービスを作成します。

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

3. 外部 IP のサービスを確認します。たとえば、**myservice** という名前のサービスを確認します。

```
$ oc get svc myservice
```

LoadBalancer タイプのサービスに外部 IP が割り当てられている場合、出力には IP が表示されます。

```
NAME        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE
myservice   172.30.74.106   172.29.0.1    3306/TCP    30s
```

27.4. 開発またはテスト目的での INGRESS CIDR のルーティング

ingress CIDR のトラフィックをクラスターのノードに送信する静的ルートを追加します。以下に例を示します。

```
# route add -net 172.29.0.0/16 gw 10.66.140.17 eth0
```

上記の例では、**172.29.0.0/16** は **ingressIPNetworkCIDR**、**10.66.140.17** はノード IP です。

27.4.1. サービス externalIP

クラスターの内部 IP アドレスに加えて、アプリケーション開発者はクラスターの外部にある IP アドレスを設定することができます。OpenShift Container Platform 管理者は、トラフィックがこの IP を持つノードに到達することを確認する必要があります。

externalIP は、master-config.yaml ファイルで設定される **externalIPNetworkCIDRs** 範囲から管理者によって選択される必要があります。**master-config.yaml** が変更される際に、マスターサービスは再起動される必要があります。

```
# master-restart api
# master-restart controllers
```

externalIPNetworkCIDR /etc/origin/master/master-config.yaml のサンプル

```
networkConfig:
  externalIPNetworkCIDR: 172.47.0.0/24
```

サービス externalIP 定義 (JSON)

```
{
    "kind": "Service",
    "apiVersion": "v1",
    "metadata": {
```

```
        "name": "my-service"
    },
    "spec": {
        "selector": {
            "app": "MyApp"
        },
        "ports": [
            {
                "name": "http",
                "protocol": "TCP",
                "port": 80,
                "targetPort": 9376
            }
        ],
        "externalIPs" : [
            "80.11.12.10"          (1)
        ]
    }
}
```

(1) ポート が公開される外部 IP アドレスの一覧です。(内部 IP アドレスに追加)

第28章 ルーターのモニターリングおよびデバッグ

28.1. 概要

基礎となる実装によっては、実行中の ルーター を複数の方法でモニターすることができます。このトピックでは、HAProxy テンプレートルーターおよびその正常性を確認するためのコンポーネントについて説明します。

28.2. 統計の表示

HAProxy ルーターは、HAProxy 統計の web リスナーを公開します。ルーターのパブリック IP アドレスと適切に設定されたポート (デフォルトは 1936) を入力して統計ページを表示し、プロンプトが出されたら管理者パスワードを入力します。このパスワードおよびポートはルーターのインストール時に設定されますが、それらはコンテナーの **haproxy.config** ファイルを表示して確認することができます。

Prometheus 形式で未加工統計を抽出するには、以下を実行します。

```
$ curl -u <user>:<password> -kv <router_IP>:<STATS_PORT>/metrics
```

このコマンドに必要な情報の取得に関する詳細は、ルーターメトリックの公開 を参照してください。

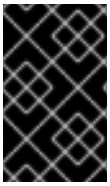
28.3. 統計ビューの無効化

デフォルトで、HAProxy 表示はポート 1936 で公開されます (パスワードで保護されたアカウントを使用する)。HAProxy 統計の公開を無効にするには、統計ポート番号として 0 を指定します。

```
$ oc adm router hap --service-account=router --stats-port=0
```

注: HAProxy は依然として統計を収集し、保存しますが、web リスナー経由での統計の 公開 が行われなくなります。要求を HAProxy ルーターコンテナー内の HAProxy AF_UNIX ソケットに送信すれば、依然として統計にアクセスできます。

```
$ cmd="echo 'show stat' | socat - UNIX-CONNECT:/var/lib/haproxy/run/haproxy.sock"
$ routerPod=$(oc get pods --selector="router=router"  \
    --template="{{with index .items 0}}{{.metadata.name}}{{end}}")
$ oc exec $routerPod -- bash -c "$cmd"
```



重要

セキュリティー保護の目的 により、**oc exec** コマンドは、特権付きコンテナーにアクセスする場合には機能しません。その代わりに、ノードホストに対して SSH を実行して必要なコンテナーで **docker exec** コマンドを使用することができます。

28.4. ログの表示

ルーターのログを表示するには、Pod で **oc logs** コマンドを実行します。ルーターは基礎となる実装を管理するプラグインプロセスとして実行されているため、このログは実際の HAProxy ログではなく、プラグインのログになります。

HAProxy で生成されるログを表示するには、以下の環境変数を使用して syslog サーバーを起動し、その位置情報をルーター Pod に渡します。

表28.1 ルーター Syslog 変数

環境変数	説明
ROUTER_SYSLOG_ADDRESS	syslog サーバーの IP アドレスです。ポートが指定されていない場合、ポート **514** がデフォルトになります。
ROUTER_LOG_LEVEL	オプション。HAProxy ログレベルを変更するように設定します。設定されていない場合は、デフォルトのログレベルは **警告** になります。これは HAProxy がサポートするログレベルに変更することができます。
ROUTER_SYSLOG_FORMAT	オプション。カスタマイズされた HAProxy ログ形式を定義するように設定します。これは HAProxy が受け入れるログ形式の文字列に変更できます。

メッセージを syslog サーバーに送信できるように実行中のルーター Pod を設定するには、以下を実行します。

```
$ oc set env dc/router ROUTER_SYSLOG_ADDRESS=<dest_ip:dest_port>
ROUTER_LOG_LEVEL=<level>
```

たとえば、以下はデフォルトポート **514** で 127.0.0.1 にログを送信するよう HAProxy を設定し、ログレベルを **debug** に変更します。

```
$ oc set env dc/router ROUTER_SYSLOG_ADDRESS=127.0.0.1 ROUTER_LOG_LEVEL=debug
```

28.5. ルーター内部の表示

routes.json

ルートは HAProxy ルーターで処理され、メモリー、ディスクおよび HAProxy 設定ファイルに保存されます。HAProxy 設定ファイルを生成するためにテンプレートに渡される内部ルート表示は **/var/lib/haproxy/router/routes.json** ファイルで確認できます。ルーティングの問題のトラブルシューティング時には、このファイルを表示して設定を有効にするために使用されているデータを確認できます。

HAProxy 設定

HAProxy 設定および特定ルート用に作成されたバックエンドは **/var/lib/haproxy/conf/haproxy.config** ファイルで確認することができます。マッピングファイルは同じディレクトリーにあります。ヘルパーのフロントエンドとバックエンドは、着信要求のバックエンドへのマッピング時にマッピングファイルを使用します。

証明書

証明書は 2 つの場所に保存されます。

- edge termination および re-encrypt 終端ルートの証明書は **/var/lib/haproxy/router/certs** ディレクトリーに保存されます。
- re-encrypt 終端ルートのバックエンドへの接続に使用される証明書は **/var/lib/haproxy/router/cacerts** ディレクトリーに保存されます。

ファイルはルートの namespace および名前で指定されます。キー、証明書および CA 証明書は単一ファイルに連結されます。OpenSSL を使用して、これらのファイルの内容を表示できます。

第29章 高可用性

29.1. 概要

このトピックでは、OpenShift Container Platform クラスターの Pod およびサービスの高可用性の設定について説明します。

IP フェイルオーバーは、ノードセットの仮想 IP (VIP) アドレスのプールを管理します。セットのすべての VIP はセットから選択されるノードによって提供されます。VIP は単一ノードが利用可能である限り提供されます。ノード上で VIP を明示的に配布する方法がないため、VIP のないノードがある可能性も、多数の VIP を持つノードがある可能性もあります。ノードが 1 つのみ存在する場合は、すべての VIP がそのノードに配置されます。

注記

VIP はクラスター外からルーティングできる必要があります。

IP フェイルオーバーは各 VIP のポートをモニターし、ポートがノードで到達可能かどうかを判別します。ポートが到達不能な場合、VIP はノードに割り当てられません。ポートが **0** に設定されている場合、このチェックは抑制されます。 check スクリプト は必要なテストを実行します。

IP フェイルオーバーは Keepalived を使用して、一連のホストで外部からアクセスできる VIP アドレスのセットをホストします。各 VIP は 1 度に 1 つのホストによって提供されます。 **Keepalived** は VRRP プロトコルを使用して (一連のホストの) どのホストがどの VIP を提供するかを判別します。ホストが利用不可の場合や **Keepalived** が監視しているサービスが応答しない場合は、VIP は一連のホストの内の別のホストに切り換えられます。したがって、VIP はホストが利用可能である限り常に提供されます。

Keepalived を実行するホストが **check** スクリプトを渡す場合、ホストは プリエンプションストラテジー に応じて、その優先順位および現在の **MASTER** の優先順位に基づいて **MASTER** 状態になります。

管理者は、状態が変更されるたびに呼び出されるスクリプトを **--notify-script=** オプションを使用して提供できます。**Keepalived** は VIP を提供する場合は **MASTER** 状態に、別のノードが VIP を提供する場合は **BACKUP** 状態に、または **check** スクリプトが失敗する場合は **FAULT** 状態になります。notify スクリプト は、状態が変更されるたびに新規の状態で呼び出されます。

OpenShift Container Platform は、**oc adm ipfailover** コマンドの実行による IP フェイルオーバーのデプロイメント設定の作成をサポートします。IP フェイルオーバーのデプロイメント設定は VIP アドレスのセットを指定し、それらの提供先となるノードのセットを指定します。クラスターには複数の IP フェイルオーバーのデプロイメント設定を持たせることができ、それぞれが固有な VIP アドレスの独自のセットを管理します。IP フェイルオーバー設定の各ノードは IP フェイルオーバー Pod として実行され、この Pod は **Keepalived** を実行します。

VIP を使用してホストネットワーク (例: ルーター) を持つ Pod にアクセスする場合、アプリケーション Pod は ipfailover Pod を実行しているすべてのノードで実行されている必要があります。これにより、いずれの ipfailover ノードもマスターになり、必要時に VIP を提供することができます。アプリケーション Pod が ipfailover のすべてのノードで実行されていない場合、一部の ipfailover ノードが VIP を提供できないか、または一部のアプリケーション Pod がトラフィックを受信できなくなります。この不一致を防ぐために、ipfailover とアプリケーション Pod の両方に同じセレクターとレプリケーション数を使用します。

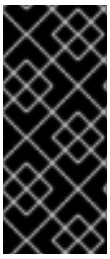
VIP を使用してサービスにアクセスする場合には、いずれのノードもノードの ipfailover セットに入れることができます。それは、(アプリケーション Pod が実行されている場所を問わず) サービスはすべてのノードで到達可能であるためです。ipfailover ノードのいずれもいつでもマスターにすることがで

きます。 サービスは外部 IP およびサービスポートを使用するか、または nodePort を使用することができます。

サービス定義で外部 IP を使用する場合、VIP は外部 IP に設定され、ipfailover のモニターポートはサービスポートに設定されます。nodePort はクラスターのすべてのノードで開かれ、サービスは VIP をサポートしているいずれのノードからのトラフィックについても負荷分散を行います。この場合、ipfailover のモニターノードはサービス定義で nodePort に設定されます。

重要

nodePort のセットアップは特権付きの操作で実行されます。



重要

サービス VIP の可用性が高い場合でも、パフォーマンスに影響が出る可能性があります。**keepalived** は、各 VIP が設定内の一部のノードによってサービスされることを確認し、他のノードに VIP がない場合でも、複数の VIP が同じノードに配置される可能性があります。ipfailover によって複数の VIP が同じノードに配置されると、VIP のセット全体で外部から負荷分散されるストラテジーが妨げられる可能性があります。

ingressIP を使用する場合は、ipfailover を ingressIP 範囲と同じ VIP 範囲を持つように設定できます。また、モニターポートを無効にすることもできます。この場合、すべての VIP がクラスター内の同じノードに表示されます。すべてのユーザーが ingressIP でサービスをセットアップし、これを高い可用性のあるサービスにすることができます。

重要

クラスター内の VIP の最大数は 255 です。

29.2. IP フェイルオーバーの設定

oc adm ipfailover コマンドを適切な オプション と共に使用し、ipfailover デプロイメント設定を作成します。

重要

現時点で、ipfailover はクラウドインフラストラクチャーと互換性がありません。AWS の場合、AWS コンソールの使用 により、Elastic Load Balancer (ELB) を使用して OpenShift Container Platform の高可用性を維持することができます。

管理者は、クラスター全体に ipfailover を設定することも、ラベルセレクターの定義に基づいてノードのサブセットに ipfailover を設定することもできます。また、複数の IP フェイルオーバーのデプロイメント設定をクラスター内に設定することもでき、それぞれの設定をクラスター内で相互に切り離すことができます。**oc adm ipfailover** コマンドは ipfailover デプロイメント設定を作成し、これによりフェイルオーバー Pod が使用される制約またはラベルに一致する各ノードで実行されるようにします。この Pod は、すべての **Keepalived** デーモン間で VRRP (Virtual Router Redundancy Protocol) を使用する Keepalived を実行し、監視されるポートでサービスが利用可能であることを確認し、利用可能でない場合は **Keepalived** が仮想 IP (VIP) を自動的に浮動させます。

実稼働環境で使用する場合には、2 つ以上のノードで **--selector=<label>** を使用してノードを選択するようにします。また、指定のラベルが付けられたセレクターのノード数に一致する **--replicas=<n>** 値を設定します。

oc adm ipfailover コマンドには、Keepalived を制御する環境変数を設定するコマンドラインオプションが含まれます。環境変数 は、**OPENSHIFT_HA_*** で開始され、必要に応じて変更できます。

たとえば、以下のコマンドは **router=us-west-ha** のラベルが付けられたノードのセレクションに対して IP フェイルオーバー設定を作成します (7 仮想 IP を持つ 4 ノードで、ルータープロセスなどのポート 80 でリッスンするサービスをモニターリング)。

```
$ oc adm ipfailover --selector="router=us-west-ha" \
    --virtual-ips="1.2.3.4,10.1.1.100-104,5.6.7.8" \
    --watch-port=80 --replicas=4 --create
```

29.2.1. 仮想 IP アドレス

Keepalived は一連の仮想 IP アドレス (VIP) を管理します。管理者はこれらすべてのアドレスについて以下の点を確認する必要があります。

- 仮想 IP アドレスは設定されたホストでクラスター外からアクセスできる。
- 仮想 IP アドレスはクラスター内でこれ以外の目的で使用されていない。

各ノードの Keepalived は、必要とされるサービスが実行中であるかどうかを判別します。実行中の場合、VIP がサポートされ、Keepalived はネゴシエーションに参加してそのノードが VIP を提供するかを決定します。これに参加するノードについては、このサービスが VIP の監視 ポートでリッスンしている、またはチェックが無効にされている必要があります。

注記

セット内の各 VIP は最終的に別のノードによって提供される可能性があります。

29.2.2. チェックおよび通知スクリプト

Keepalived は、オプションのユーザー指定のチェックスクリプトを定期的に実行してアプリケーションの正常性をモニターします。たとえば、このスクリプトは要求を発行し、応答を検証することで web サーバーをテストします。

スクリプトは **oc adm ipfailover** コマンドに **--check-script=<script>** オプションを指定して実行されます。このスクリプトは **PASS** の場合は **0** で終了するか、または **FAIL** の場合は **1** で終了する必要があります。

デフォルトでチェックは 2 秒ごとに実行されますが、**--check-interval=<seconds>** オプションを使用して頻度を変更することができます。

チェックスクリプトが指定されない場合、TCP 接続 をテストする単純なデフォルトスクリプトが実行されます。このデフォルトテストは、モニターポートが **0** の場合は抑制されます。

それぞれの仮想 IP (VIP) について、keepalived はノードの状態を保持します。ノードの VIP は **MASTER**、**BACKUP**、または **FAULT** の状態になります。**FAULT** 状態にないノードのすべての VIP はネゴシエーションに参加し、VIP の **MASTER** を決定します。選ばれなかったすべてのノードは **BACKUP** 状態になります。**MASTER** での **check** スクリプトが失敗すると、VIP は **FAULT** 状態になり、再ネゴシエーションがトリガーされます。**BACKUP** が失敗すると、VIP は **FAULT** 状態になります。**check** スクリプトが **FAULT** 状態の VIP に再度渡されると、その VIP は **FAULT** 状態を終了して **MASTER** のネゴシエーションを行います。結果としてその VIP の状態は **MASTER** または **BACKUP** のいずれかになります。

管理者はオプションの **notify** スクリプトを提供できます。このスクリプトは状態が変更されるたびに呼び出されます。 **Keepalived** は以下の 3 つのパラメーターをこのスクリプトに渡します。

- **$1** - "GROUP"|"INSTANCE"
- **$2**: グループまたはインスタンスの名前です。
- **$3**: 新規の状態 ("MASTER"|"BACKUP"|"FAULT") です。

これらのスクリプトは IP フェイルオーバー Pod で実行され、ホストのファイルシステムではなく Pod のファイルシステムを使用します。オプションにはスクリプトへの完全パスが必要です。管理者は Pod でスクリプトを利用可能にし、**notify** スクリプトを実行して結果を抽出できるようにする必要があります。スクリプトを提供する方法として、ConfigMap の使用が推奨されます。

check および **notify** スクリプトの完全パス名は、**keepalived** 設定ファイル、**/etc/keepalived/keepalived.conf** に追加されます。これは **keepalived** が起動するたびに読み込まれます。スクリプトは以下のように ConfigMap を使って Pod に追加できます。

1. 必要なスクリプトを作成し、これを保持する ConfigMap を作成します。スクリプトには入力引数は指定されず、**OK** の場合は **0** を、**FAIL** の場合は **1** を返します。
   check スクリプト **mycheckscript.sh**:

```
#!/bin/bash
    # Whatever tests are needed
    # E.g., send request and verify response
exit 0
```

2. ConfigMap を作成します。

```
$ oc create configmap mycustomcheck --from-file=mycheckscript.sh
```

3. スクリプトを Pod に追加する方法として、**oc** コマンドの使用またはデプロイメント設定の編集の 2 つの方法があります。どちらの場合も、マウントされた **configMap** ファイルの **defaultMode** は実行を許可する必要があります。通常は、値 **0755** (**493**、10 進数) が使用されます。

   a. **oc** コマンドの使用:

```
$ oc env dc/ipf-ha-router \
    OPENSHIFT_HA_CHECK_SCRIPT=/etc/keepalive/mycheckscript.sh
$ oc set volume dc/ipf-ha-router --add --overwrite \
    --name=config-volume \
    --mount-path=/etc/keepalive \
    --source='{"configMap": { "name": "mycustomcheck", "defaultMode": 493}}'
```

   b. **ipf-ha-router** デプロイメント設定の編集:

      i. **oc edit dc ipf-ha-router** を使用し、テキストエディターでルーターデプロイメント設定を編集します。

```
...
    spec:
      containers:
      - env:
        - name: OPENSHIFT_HA_CHECK_SCRIPT (1)
```

```
        value: /etc/keepalive/mycheckscript.sh
...
      volumeMounts: (2)
      - mountPath: /etc/keepalive
        name: config-volume
    dnsPolicy: ClusterFirst
...
    volumes: (3)
    - configMap:
        defaultMode: 0755 (4)
        name: customrouter
      name: config-volume
...
```

(1) **spec.container.env** フィールドで、マウントされたスクリプトファイルを参照する **OPENSHIFT_HA_CHECK_SCRIPT** 環境変数を追加します。

(2) **spec.container.volumeMounts** フィールドを追加してマウントポイントを作成します。

(3) 新規の **spec.volumes** フィールドを追加して ConfigMap に言及します。

(4) これはファイルの実行パーミッションを設定します。読み取られる場合は 10 進数 (**493**) で表示されます。

ii. 変更を保存し、エディターを終了します。これにより **ipf-ha-router** が再起動します。

29.2.3. VRRP プリエンプション

ホストが check スクリプトを渡すことで **FAULT** 状態を終了する場合、その新規ホストが現在の **MASTER** 状態にあるホストよりも優先度が低い場合は **BACKUP** になります。ただしそのホストの優先度が高い場合は、プリエンプションストラテジーがクラスター内でのそのロールを決定します。

nopreempt ストラテジーは **MASTER** を低優先度のホストから高優先度のホストに移行しません。デフォルトの **preempt 300** の場合、**keepalived** は指定された 300 秒の間待機し、**MASTER** を優先度の高いホストに移行します。

プリエンプションを指定するには、以下を実行します。

a. **preemption-strategy** を使用して ipfailover を作成します。

```
$ oc adm ipfailover --preempt-strategy=nopreempt \
  ...
```

b. **oc set env** コマンドを使用して変数を設定します。

```
$ oc set env dc/ipf-ha-router \
   --overwrite=true \
   OPENSHIFT_HA_PREEMPTION=nopreempt
```

c. **oc edit dc ipf-ha-router** を使用してルーターデプロイメント設定を編集します。

```
...
   spec:
```

```
      containers:
      - env:
        - name: OPENSHIFT_HA_PREEMPTION (1)
          value: nopreempt
...
```

29.2.4. Keepalived マルチキャスト

OpenShift Container Platform の IP フェイルオーバーは **keepalived** を内部で使用します。

重要

前述のラベルが付いたノードで **multicast** が有効にされており、それらが 224.0.0.18 (VRRP マルチキャスト IP アドレス) のネットワークトラフィックを許可することを確認します。

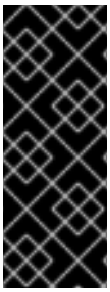
keepalived デーモンを起動する前に、起動スクリプトは、マルチキャストトラフィックのフローを許可する **iptables** ルールを検証します。このルールがない場合、起動スクリプトは新規ルールを作成し、これを IP テーブル接続に追加します。この新規ルールが IP テーブルに追加される場所は **--iptables-chain=** オプションによって異なります。**--iptables-chain=** オプションが指定される場合、ルールはオプションで指定されるチェーンに追加されます。そうでない場合は、ルールは **INPUT** チェーンに追加されます。

重要

iptables ルールは、1 つ以上の **keepalived** デーモンがノードで実行されている場合に存在している必要があります。

iptables ルールは、最後の **keepalived** デーモンの終了後に削除できます。このルールは自動的に削除されません。

各ノードで **iptables** ルールを手動で管理できます。(ipfailover が **--iptable-chain=""** オプションで作成されていない限り) 何も存在しない場合にこのルールが作成されます。



重要

手動で追加されたルールがシステム起動後も保持されることを確認する必要があります。

すべての **keepalived** デーモンはマルチキャスト 224.0.0.18 で VRRP を使用してそのピアとネゴシエーションするので注意が必要です。それぞれの VIP に異なる VRRP-id (0..255 の範囲) が必要です。

```
$ for node in openshift-node-{5,6,7,8,9}; do   ssh $node <<EOF

export interface=${interface:-"eth0"}
echo "Check multicast enabled ... ";
ip addr show $interface | grep -i MULTICAST

echo "Check multicast groups ... "
ip maddr show $interface | grep 224.0.0
```

```
EOF
done;
```

29.2.5. コマンドラインオプションおよび環境変数

表29.1 コマンドラインオプションおよび環境変数

オプション	変数名	デフォルト	注記
--watch-port	**OPENSHIFT_HA_MONITOR_PORT**	80	ipfailover Pod は、各 VIP のこのポートに対して TCP 接続を開こうとします。接続が設定されると、サービスは実行中であると見なされます。このポートが 0 に設定される場合、テストは常にパスします。
--interface	**OPENSHIFT_HA_NETWORK_INTERFACE**		使用する ipfailover のインターフェイス名で、VRRP トラフィックを送信するために使用されます。デフォルトで **eth0** が使用されます。
--replicas	**OPENSHIFT_HA_REPLICA_COUNT**	2	作成するレプリカの数。これは、ipfailover デプロイメント設定の **spec.replicas** 値に一致している必要があります。
--virtual-ips	**OPENSHIFT_HA_VIRTUAL_IPS**		複製する IP アドレス範囲の一覧です。これは指定する必要があります(例: 1.2.3.4-6,1.2.3.9.)。 詳細は、こちら を参照してください。
--vrrp-id-offset	**OPENSHIFT_HA_VRRP_ID_OFFSET**	0	詳細は、VRRP ID オフセット を参照してください。
--virtual-ip-groups	**OPENSHIFT_HA_VIP_GROUPS**		VRRP に作成するグループの数です。これが設定されていない場合、グループは **--virtual-ips** オプションで指定されている仮想 IP 範囲ごとに作成されます。詳細は、254 を超えるアドレスの IP フェイルオーバーの設定 を参照してください。
--iptables-chain	**OPENSHIFT_HA_IPTABLES_CHAIN**	INPUT	iptables チェーンの名前であり、**iptables** ルールを自動的に追加し、VRRP トラフィックをオンにすることを許可するために使用されます。この値が設定されていない場合、**iptables** ルールは追加されません。チェーンが存在しない場合は作成されません。

オプション	変数名	デフォルト	注記
--check-script	**OPENSHIFT_HA_CHECK_SCRIPT**		Pod のファイルシステム内の、アプリケーションの動作を確認するために定期的に実行されるスクリプトの完全パス名です。詳細は、こちら を参照してください。
--check-interval	**OPENSHIFT_HA_CHECK_INTERVAL**	2	check スクリプトが実行される期間 (秒単位) です。
--notify-script	**OPENSHIFT_HA_NOTIFY_SCRIPT**		Pod ファイルシステム内の、状態が変更されるたびに実行されるスクリプトの完全パス名です。詳細は、こちら を参照してください。
--preemption-strategy	**OPENSHIFT_HA_PREEMPTION**	preempt_300	新たな優先度の高いホストを処理するためのストラテジーです。詳細は、VRRP プリエンプション のセクションを参照してください。

29.2.6. VRRP ID オフセット

ipfailover デプロイメント設定で管理される各 ipfailover Pod (ノード/レプリカあたり 1 Pod) は **keepalived** デーモンを実行します。作成される ipfailover デプロイメント設定が多くなると、作成される Pod も多くなり、共通の VRRP ネゴシエーションに参加するデーモンも多くなります。このネゴシエーションはすべての **keepalived** デーモンによって実行され、これはどのノードがどの仮想 IP (VIP) を提供するかを決定します。

keepalived は内部で固有の vrrp-id を各 VIP に割り当てます。ネゴシエーションはこの vrrp-id セットを使用し、決定後には優先される vrrp-id に対応する VIP が優先されるノードで提供されます。

したがって、ipfailover デプロイメント設定で定義されるすべての VIP について、ipfailover Pod は対応する vrrp-id を割り当てます。これは、**--vrrp-id-offset** から開始し、順序に従って vrrp-id を VIP の一覧に割り当てることによって実行されます。vrrp-id には範囲 1..255 の値を設定できます。

複数の ipfailover デプロイメント設定がある場合、デプロイメント設定の VIP 数を増やす余地があることや vrrp-id 範囲のいずれも重複しないことを確認できるよう **--vrrp-id-offset** を注意して指定する必要があります。

29.2.7. 254 を超えるアドレスについての IP フェイルオーバーの設定

IP フェイルオーバー管理は VIP アドレスの 254 グループに制限されています。デフォルトでは、OpenShift Container Platform は各グループに 1 つの IP アドレスを割り当てます。**virtual-ip-groups** オプションを使用するとこれを変更でき、IP フェイルオーバーの設定 時に複数の IP アドレスをそれぞれのグループに配置し、各 VRRP インスタンスに利用できる VIP グループの数を定義できます。

VIP の作成により、VRRP フェイルオーバーの発生時の広範囲の VRRP の割り当てが作成され、これはクラスター内のすべてのホストがローカルにサービスにアクセスする場合に役立ちます。たとえば、サービスが **ExternalIP** で公開されている場合などがこれに含まれます。

注記

フェイルオーバーのルールとして、ルーターなどのサービスは特定の 1 つのホストに制限しません。代わりに、サービスは、IP フェイルオーバーの発生時にサービスが新規ホストに再作成されないように各ホストに複製可能な状態にする必要があります。

注記

OpenShift Container Platform のヘルスチェックを使用している場合、IP フェイルオーバーおよびグループの性質上、グループ内のすべてのインスタンスはチェックされません。そのため、Kubernetes ヘルスチェック を使ってサービスが有効であることを確認する必要があります。

```
# oc adm ipfailover <ipfailover_name> --create \
    ...
    --virtual-ip-groups=<number_of_ipfailover_groups>
```

たとえば、7 つの VIP のある環境で **--virtual-ip-groups** が **3** に設定されている場合、これは 3 つのグループを作成し、3 つの VIP を最初のグループに、2 つの VIP を 2 つの残りのグループにそれぞれ割り当てます。

注記

--virtual-ip-groups オプションで設定されるグループの数がフェイルオーバーに設定される IP アドレスの数よりも小さい場合、グループには複数の IP アドレスが含まれ、アドレスのすべてが単一のユニットとして移行します。

29.2.8. 高可用サービスの設定

以下の例では、ノードのセットに IP フェイルオーバーを指定して可用性の高い **router** および **geo-cache** ネットワークサービスをセットアップする方法について説明します。

1. サービスに使用されるノードにラベルを付けます。の手順は、OpenShift Container Platform クラスターのすべてのノードでサービスを実行し、クラスターのすべてのノード内で固定されない VIP を使用する場合はオプションになります。
   以下の例では、US west の地理的区分でトラフィックを提供するノードのラベルを定義します (**ha-svc-nodes=geo-us-west**)。

   ```
   $ oc label nodes openshift-node-{5,6,7,8,9} "ha-svc-nodes=geo-us-west"
   ```

2. サービスアカウントを作成します。ipfailover を使用したり、(環境ポリシーによって異なる) ルーターを使用する場合は事前に作成された **router** サービスアカウントか、または新規の ipfailover サービスアカウントのいずれかを再利用できます。
   以下の例は、デフォルト namespace で ipfailover という名前の新規サービスアカウントを作成します。

   ```
   $ oc create serviceaccount ipfailover -n default
   ```

3. デフォルト namespace の ipfailover サービスアカウントを **privileged** SCC に追加します。

   ```
   $ oc adm policy add-scc-to-user privileged system:serviceaccount:default:ipfailover
   ```

4. **router** および **geo-cache** サービスを起動します。

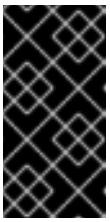
重要

ipfailover は手順 1 のすべてのノードで実行されるため、手順 1 のすべてのノードでルーター/サービスを実行することも推奨されます。

a. 最初の手順で使用されるラベルに一致するノードでルーターを起動します。以下の例では、ipfailover サービスアカウントを使用して 5 つのインスタンスを実行します。

```
$ oc adm router ha-router-us-west --replicas=5 \
    --selector="ha-svc-nodes=geo-us-west" \
    --labels="ha-svc-nodes=geo-us-west" \
    --service-account=ipfailover
```

b. 各ノードでレプリカと共に **geo-cache** サービスを実行します。**geo-cache** サービスの実行については、設定サンプル を参照してください。

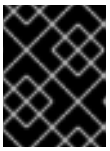


重要

ファイルで参照される **myimages/geo-cache** コンテナーイメージが使用するイメージに置き換えられていることを確認します。レプリカの数を **geo-cache** ラベルのノード数に変更します。ラベルが最初の手順で使用したものと一致していることを確認します。

```
$ oc create -n <namespace> -f ./examples/geo-cache.json
```

5. **router** および **geo-cache** サービスの ipfailover を設定します。それぞれに独自の VIP があり、いずれも最初の手順の **ha-svc-nodes=geo-us-west** のラベルが付けられた同じノードを使用します。レプリカの数が最初の手順のラベル設定に一覧表示されているノード数と一致していることを確認してください。



重要

router、**geo-cache** および ipfailover はすべてデプロイメント設定を作成します。 それらの名前はすべて異なっている必要があります。

6. ipfailover が必要なインスタンスでモニターする必要のある VIP およびポート番号を指定します。
router の ipfailover コマンド:

```
$ oc adm ipfailover ipf-ha-router-us-west \
    --replicas=5 --watch-port=80 \
    --selector="ha-svc-nodes=geo-us-west" \
    --virtual-ips="10.245.2.101-105" \
    --iptables-chain="INPUT" \
    --service-account=ipfailover --create
```

以下は、ポート 9736 でリッスンする **geo-cache** サービスの **oc adm ipfailover** コマンドです。2 つの **ipfailover** デプロイメント設定があるため、それぞれの VIP が独自のオフセットを取得できるように **--vrrp-id-offset** を設定する必要があります。この場合、**10** の値を設定すると、**ipf-ha-geo-cache** が 10 から開始するために **ipf-ha-router-us-west** には最大 10 の VIP (0-9) を持たせることができることを意味します。

```
$ oc adm ipfailover ipf-ha-geo-cache \
    --replicas=5 --watch-port=9736 \
    --selector="ha-svc-nodes=geo-us-west" \
    --virtual-ips=10.245.3.101-105 \
    --vrrp-id-offset=10 \
    --service-account=ipfailover --create
```

上記のコマンドでは、各ノードに **ipfailover**、**router**、および **geo-cache** Pod があります。各 ipfailover 設定の VIP のセットは重複してななら ず、外部またはクラウド環境の別の場所で使用することはできません。それぞれの例の 5 つの VIP **10.245.{2,3}.101-105** は、2 つの ipfailover デプロイメント設定で提供されます。IP フェイルオーバーはどのアドレスがどのノードで提供されるかを動的に選択します。

管理者は、すべての **router** VIP が同じ **router** を参照し、すべての **geo-cache** VIP が同じ **geo-cache** サービスを参照することを前提とした上で VIP アドレスを参照する外部 DNS をセットアップします。1 つのノードが実行中である限り、すべての VIP アドレスが提供されます。

29.2.8.1. IP フェイルオーバー Pod のデプロイ

postgresql-ingress サービスの定義に基づいてノードポート 32439 および外部 IP アドレスでリッスンする postgresql をモニターするために ipfailover ルーターをデプロイします。

```
$ oc adm ipfailover ipf-ha-postgresql \
    --replicas=1 \ 1
    --selector="app-type=postgresql" \ 2
    --virtual-ips=10.9.54.100 \ 3
    --watch-port=32439 \ 4
    --service-account=ipfailover --create
```

1 1 デプロイするインスタンスの数を指定します。

2 ipfailover がデプロイされる場所を制限します。

3 モニターする仮想 IP アドレスです。

4 各ノード上の ipfailover がモニターするポートです。

29.2.9. 高可用サービスの仮想 IP の動的更新

IP フェイルオーバーのデフォルトのデプロイメント方法として、デプロイメントを再作成します。高可用のルーティングサービスの動的更新を最小限のダウンタイムまたはダウンタイムなしで実行するには、以下を実行する必要があります。

- ローリング更新 (Rolling Update) ストラテジーを使用するように IP フェイルオーバーサービスデプロイメント設定を更新する。
- 仮想 IP アドレスの更新された一覧またはセットを使用して **OPENSHIFT_HA_VIRTUAL_IPS** 環境変数を更新します。

以下の例は、デプロイメントストラテジーおよび仮想 IP アドレスを動的に更新する方法について示しています。

1. 以下を使用して作成された IP フェイルオーバー設定を見てみましょう。

```
$ oc adm ipfailover ipf-ha-router-us-west \
    --replicas=5 --watch-port=80 \
    --selector="ha-svc-nodes=geo-us-west" \
    --virtual-ips="10.245.2.101-105" \
    --service-account=ipfailover --create
```

2. デプロイメント設定を編集します。

```
$ oc edit dc/ipf-ha-router-us-west
```

3. **spec.strategy.type** フィールドを **Recreate** から **Rolling** に更新します。

```
spec:
  replicas: 5
  selector:
    ha-svc-nodes: geo-us-west
  strategy:
    resources: {}
    rollingParams:
      maxSurge: 0
    type: Rolling (1)
```

(1) **Rolling** に設定します。

4. 追加の仮想 IP アドレスを含めるように **OPENSHIFT_HA_VIRTUAL_IPS** 環境変数を更新します。

```
- name: OPENSHIFT_HA_VIRTUAL_IPS
  value: 10.245.2.101-105,10.245.2.110,10.245.2.201-205 (1)
```

(1) **10.245.2.110,10.245.2.201-205** が一覧に追加されます。

5. VIP のセットに一致するよう外部 DNS を更新します。

29.3. サービスの EXTERNALIP および NODEPORT の設定

ユーザーは VIP をサービスの ExternalIP として割り当てることができます。**Keepalived** は、各 VIP が ipfailover 設定のノードで提供されることを確認します。要求がそのノードに到達すると、クラスター内のすべてのノードで実行されているサービスがサービスのエンドポイント間で要求の負荷の分散を行います。

NodePorts を ipfailover 監視ポートに設定して、**keepalived** がアプリケーションが実行中であることを確認できるようにします。NodePort はクラスター内のすべてのノードで公開されるため、すべての ipfailover ノードの **keepalived** で利用可能になります。

29.4. INGRESSIP の高可用性

クラウド以外のクラスターでは、ipfailover およびサービスへの ingressIP を組み合わせることができます。結果として、ingressIP を使用してサービスを作成するユーザーに高可用サービスが提供されます。

この方法では、まず **ingressIPNetworkCIDR** 範囲を指定し、次に ipfailover 設定を作成する際に同じ範囲を使用します。

ipfailover はクラスター全体に対して最大 255 の VIP をサポートするため、**ingressIPNetworkCIDR** は **/24** 以下に設定する必要があります。

第30章 IPTABLES

30.1. 概要

システムコンポーネントには、OpenShift Container Platform、コンテナー、および適切なネットワーク操作のためにカーネルの iptables 設定に依存するファイアウォールポリシーを管理するソフトウェアなど、数多くのコンポーネントがあります。さらに、クラスター内のすべてのノードの iptables 設定はネットワークが機能するように正しくなければなりません。

すべてのコンポーネントは、他のコンポーネントが iptables をどのように使用するかを認識せずに 独立して iptables を使用します。そのため、あるコンポーネントを別のコンポーネントの設定から分離することが容易になります。さらに、OpenShift Container Platform および Docker サービスは、iptables がそれらがセットアップした時と全く同じ設定であると仮定します。それらは他のコンポーネントによって導入される変更を検出しない場合がありますが、これらを検出する場合は修正の実装により一部の遅れが生じる可能性があります。特に、OpenShift Container Platform は問題を監視し、解決します。ただし、Docker サービスはそうではありません。

重要

ノード上の iptables 設定に対して加えるいかなる変更も OpenShift Container Platform および Docker サービスの操作に影響を与えないものであることを確認してください。また多くの場合、変更はクラスター内のすべてのノードに対して実行される必要があります。iptables は複数の同時ユーザーを持つように設計されておらず、OpenShift Container Platform および Docker ネットワークに障害が発生する可能性があるため、これを変更する際には注意が必要です。

OpenShift Container Platform は複数のチェーンを提供しますが、それらの 1 つは、管理者が独自の目的で使用することが意図されている **OPENSHIFT-ADMIN-OUTPUT-RULES** です。

詳細は、外部リソースへのアクセスを制限するための iptables ルールの使用 を参照してください。

OpenShift Container Platform および Docker ネットワークが適性に機能するために、カーネル iptables のチェーン、チェーンの順序、およびルールがクラスター内の各ノードに適切に設定される必要があります。システム内には、カーネル iptables と対話し、OpenShift Container Platform および Docker サービスに意図せずに影響を与える可能性のあるツールやサービスがシステムいくつかあります。

30.2. IPTABLES

iptables ツールは、Linux カーネルの IPv4 パケットフィルターのテーブルを設定し、維持し、検査するために使用できます。

ファイアウォールなどの他の使用とは別に、OpenShift Container Platform および Docker サービスはチェーンを一部のテーブルで管理します。チェーンは特定の順序で挿入され、ルールはそれぞれのニーズに応じて固有のものになります。

注意

iptables --flush [chain] は、キーが必要な設定を削除できます。このコマンドを実行しないでください。

30.3. IPTABLES.SERVICE

iptables サービスはローカルのネットワークファイアウォールをサポートします。これは、iptables 設定を完全に制御することを想定します。これが起動すると、詳細な iptables 設定をフラッシュし、それを復元します。ルールはその設定ファイル **/etc/sysconfig/iptables** から復元されます。設定ファイルは操作時に最新の状態に保たれないため、動的に追加されたルールは毎回の再起動時に失われます。

警告

iptables.service を停止し、起動することにより、OpenShift Container Platform および Docker で必要な設定が破棄されます。OpenShift Container Platform および Docker にはこの変更は通知されません。

```
# systemctl disable iptables.service
# systemctl mask iptables.service
```

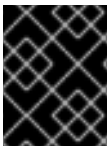
iptables.service を実行する必要がある場合、制限された設定を設定ファイルに維持し、OpenShift Container Platform および Docker を使用してそれらが必要とするルールをインストールするようにします。

iptables.service 設定は以下からロードされます。

```
/etc/sysconfig/iptables
```

ルールの永続的な変更を実行するには、このファイルで変更を編集します。Docker または OpenShift Container Platform ルールは含めないようにしてください。

iptables.service がノードで起動または再起動した後は、Docker サービスおよび **atomic-openshift-node.service** を再起動して、必要な iptables 設定を再構築する必要があります。



重要

Docker サービスの再起動により、ノードで実行されているすべてのコンテナーが停止し、再起動されます。

```
# systemctl restart iptables.service
# systemctl restart docker
# systemctl restart atomic-openshift-node.service
```

第31章 ストラテジーによるビルドのセキュリティー保護

31.1. 概要

OpenShift Container Platform の ビルド は、Docker デーモンソケットにアクセスできる 特権付きコンテナー で実行されます。セキュリティー対策として、ビルドおよびそれらのビルドに使用されるストラテジーを実行するユーザーを制限することを推奨します。カスタムビルド は、ソースビルド よりも本質的に安全性が低いと言えます。それらはノードの Docker ソケットへの完全なアクセスを持つ可能性があり、それらのアクセスでビルド内で任意のコードを実行する可能性があるためです。そのため、これはデフォルトで無効にされます。Docker ビルド のパーミッションを付与する場合についても、Docker ビルドロジックの脆弱性により権限がホストノードに付与される可能性があるために注意が必要です。

警告

Docker およびカスタムビルドの間、Docker デーモンによって実行されるアクションは特権があり、ホストネットワーク namespace で実行されます。このようなアクションは、EgressNetworkPolicy オブジェクトおよび静的 egress IP アドレスで定義されるネットワークルールを含む設定済みネットワークルールをバイパスします。

デフォルトで、ビルドを作成できるすべてのユーザーには Docker および Source-to-Image (S2I) ビルドストラテジーを使用するためのパーミッションが付与されます。**cluster-admin** 権限を持つユーザーは、このページの ユーザーへのビルドストラテジーのグローバルな制限 セクションで説明されているように、カスタムビルドストラテジーを有効にすることができます。

許可ポリシー を使用して、どのユーザーがどのビルドストラテジーを使用してビルドできるかを制限することができます。各ビルドには、対応するビルドサブリソースがあります。ストラテジーを使用してビルド作成するには、ユーザーにビルドを作成するパーミッション および ビルドストラテジーのサブリソースで作成するパーミッションがなければなりません。ビルドストラテジーのサブリソースでの **create** パーミッションを付与するデフォルトロールが提供されます。

表31.1 ビルドストラテジーのサブリソースおよびロール

ストラテジー	サブリソース	ロール
Docker	ビルド/docker	system:build-strategy-docker
Source-to-Image (S2I)	ビルド/ソース	system:build-strategy-source
カスタム	ビルド/カスタム	system:build-strategy-custom
JenkinsPipeline	ビルド/jenkinspipeline	system:build-strategy-jenkinspipeline

31.2. ビルドストラテジーのグローバルな無効化

特定のビルドストラテジーへのアクセスをグローバルに禁止するには、cluster-admin 権限を持つユーザーとしてログインし、**system:authenticated** グループから対応するロールを削除し、アノテーション **openshift.io/reconcile-protect: "true"** を適用してそれらを API の再起動間での変更から保護します。以下の例では、Docker ビルドストラテジーを無効にする方法を示します。

1. **openshift.io/reconcile-protect** アノテーションの適用

```
$ oc edit clusterrolebinding system:build-strategy-docker-binding

apiVersion: v1
groupNames:
- system:authenticated
kind: ClusterRoleBinding
metadata:
  annotations:
    openshift.io/reconcile-protect: "true" 1
  creationTimestamp: 2018-08-10T01:24:14Z
  name: system:build-strategy-docker-binding
  resourceVersion: "225"
  selfLink: /oapi/v1/clusterrolebindings/system%3Abuild-strategy-docker-binding
  uid: 17b1f3d4-9c3c-11e8-be62-0800277d20bf
roleRef:
  name: system:build-strategy-docker
subjects:
- kind: SystemGroup
  name: system:authenticated
userNames:
- system:serviceaccount:management-infra:management-admin
```

1 **openshift.io/reconcile-protect** アノテーションの値を **"true"** に変更します。デフォルトでは **"false"** に設定されています。

2. ロールを削除します。

```
$ oc adm policy remove-cluster-role-from-group system:build-strategy-docker
system:authenticated
```

3.2 よりも前のバージョンでは、ビルドストラテジーのサブリソースは **admin** および **edit** ロールに組み込まれていました。

ビルドストラテジーのサブリソースもこれらのロールから削除されることを確認します。

```
$ oc edit clusterrole admin
$ oc edit clusterrole edit
```

それぞれのロールについて、無効にするストラテジーのリソースに対応する行を削除します。

admin の Docker ビルドストラテジーの無効化

```
kind: ClusterRole
metadata:
  name: admin
...
rules:
```

```
- resources:
  - builds/custom
  - builds/docker 1
  - builds/source
  ...
...
```

1 **admin** ロールを持つユーザーに対して、Docker ビルドをグローバルに無効にするためにこの行を削除します。

31.3. ユーザーへのビルドストラテジーのグルーバルな制限

一連の特定ユーザーのみが特定のストラテジーでビルドを作成できるようにするには、以下を実行します。

1. ビルドストラテジーへのグローバルアクセスを無効にします。.
2. ビルドストラテジーに対応するロールを特定ユーザーに割り当てます。たとえば、**system:build-strategy-docker** クラスターロールをユーザー **devuser** に追加するには、以下を実行します。

```
$ oc adm policy add-cluster-role-to-user system:build-strategy-docker devuser
```

警告

ユーザーに対して **builds/docker** サブリソースへのクラスターレベルでのアクセスを付与することは、そのユーザーがビルドを作成できるすべてのプロジェクトにおいて、Docker ストラテジーを使ってビルドを作成できることを意味します。

31.4. プロジェクト内でのユーザーへのビルドストラテジーの制限

ユーザーにビルドストラテジーをグローバルに付与するのと同様に、プロジェクト内の特定ユーザーのセットのみが特定ストラテジーでビルドを作成できるようにするには、以下を実行します。

1. ビルドストラテジーへのグローバルアクセスを無効にします。.
2. ビルドストラテジーに対応するロールをプロジェクト内の特定ユーザーに付与します。たとえば、プロジェクト **devproject** 内の **system:build-strategy-docker** ロールをユーザー **devuser** に追加するには、以下を実行します。

```
$ oc adm policy add-role-to-user system:build-strategy-docker devuser -n devproject
```

第32章 SECCOMP を使用したアプリケーション機能の制限

32.1. 概要

seccomp (セキュアコンピューティングモード) は、アプリケーションが行うシステム呼び出しのセットを制限し、クラスター管理者が OpenShift Container Platform で実行されるワークロードのセキュリティーを強化するために使用されます。

seccomp サポートは Pod 設定の 2 つのアノテーションを使用して行われます。

- **seccomp.security.alpha.kubernetes.io/pod**: Pod のすべてのコンテナーに適用されるプロファイルです (上書きなし)。
- **container.seccomp.security.alpha.kubernetes.io/<container_name>**: コンテナー固有のプロファイルです (上書きあり)。

重要

デフォルトで、コンテナーは **unconfined** seccomp 設定で実行されます。

詳細な設計情報については、seccomp 設計についてのドキュメント を参照してください。

32.2. SECCOMP の有効化

seccomp は Linux カーネルの 1 つの機能です。seccomp がシステムで有効にされていることを確認するには、以下を実行します。

```
$ cat /boot/config-`uname -r` | grep CONFIG_SECCOMP=
CONFIG_SECCOMP=y
```

32.3. OPENSHIFT CONTAINER PLATFORM での SECCOMP の設定

seccomp プロファイルは json ファイルであり、システムコールを提供し、システムコールの呼び出し時に取るべき適切なアクションを実行します。

1. seccomp プロファイルを作成します。
   多くの場合は、デフォルトのプロファイル だけで十分ですが、クラスター管理者は個別システムのセキュリティー制約を定義する必要があります。

   独自のカスタムプロファイルを作成するには、**seccomp-profile-root** ディレクトリーですべてのノードのファイルを作成します。

   デフォルトの **docker/default** プロファイルを使用している場合は、これを作成する必要はありません。

2. 適切な ノード設定マップ で、**kubeletArguments** を使用し、プロファイルを保存するために **seccomp-profile-root** ディレクトリーを使用するようにノードを設定します。

```
kubeletArguments:
  seccomp-profile-root:
    - "/your/path"
```

3. 変更を適用するためにノードサービスを再起動します。

```
# systemctl restart atomic-openshift-node
```

4. 使用できるプロファイルを制御し、デフォルトプロファイルを設定するために、**seccompProfiles** フィールドで、SCC を設定 します。最初のプロファイルがデフォルトとして使用されます。
seccompProfiles フィールドで使用できる形式には以下が含まれます。

- **docker/default**: コンテナーランタイムのデフォルトプロファイルです (いずれのプロファイルも不要です)。
- **unconfined**: 拘束のないプロファイルで、seccomp を無効にします。
- **localhost/<profile-name>**: ノードのローカル seccomp プロファイルの root にインストールされるプロファイルです。
たとえば、デフォルトの **docker/default** プロファイルを使用している場合、以下で SCC を設定します。

```
seccompProfiles:
- docker/default
```

32.4. OPENSHIFT CONTAINER PLATFORM でのカスタム SECCOMP プロファイルの設定

クラスターの Pod がカスタムプロファイルで実行されるようにするには、以下を実行します。

1. **seccomp-profile-root** に seccomp プロファイルを作成します。
2. **seccomp-profile-root** を設定します。

```
kubeletArguments:
  seccomp-profile-root:
    - "/your/path"
```

3. 変更を適用するためにノードサービスを再起動します。

```
# systemctl restart atomic-openshift-node
```

4. SCC を設定します。

```
seccompProfiles:
- localhost/<profile-name>
```

第33章 SYSCTL

33.1. 概要

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 Container Platform 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に入れられます。 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。

33.2. SYSCTL について

Linux では、管理者は sysctl インターフェイ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 (共通の接頭辞: **kernel.**)
- ネットワーク (共通の接頭辞: **net.**)
- 仮想メモリー (共通の接頭辞: **vm.**)
- MDADM (共通の接頭辞: **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。すてのパラメーターの一覧を取得するには、以下を実行できます。

```
$ sudo sysctl -a
```

33.3. NAMESPACE を使用した SYSCTL VS ノードレベルの SYSCTL

現時点の 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.***

さらに、**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 の採用は、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。

システム上で namespace が使用されている **net.*** sysctl を確認するには、以下のコマンドを実行します。

```
$ podman run --rm -ti docker.io/fedora \
    /bin/sh -c "dnf install -y findutils && find /proc/sys/ \
    | grep -e /proc/sys/net"
```

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 Pod のみをそれらのノードに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。

33.4. 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。適切な namespace の設定に加えて、安全な sysctl は同じノード上の Pod 間で適切に分離する必要があります。つまり、Pod ごとに安全な sysctl を設定することについて以下の点に留意する必要があります。

- 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を取得を許可しないものである。

namespace を使用した sysctl は必ずしも常に安全であると見なされる訳ではありません。

現時点で、OpenShift Container Platform 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**

この一覧は、kubelet による分離メカニズムのサポートを強化する今後のバージョンでさらに拡張される可能性があります。

すべての安全な sysctl はデフォルトで有効にされます。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、起動に失敗します。

33.5. 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それらをノードで個別に有効にする必要があります。それらは namespace を使用した sysctl のみを有効にできます。

SCC (Security Context Constraints) の **forbiddenSysctls** および **allowedUnsafeSysctls** フィールドに sysctl または sysctl パターンの一覧を指定して、Pod に設定できる sysctl をさらに制御できます。

- **forbiddenSysctls** オプションは、特定の sysctl を除外します。
- **allowedUnsafeSysctls** オプションは、高パフォーマンスやリアルタイムのアプリケーションチューニングなどの特定ニーズを管理します。

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、またはノードの完全な破損などの深刻な問題が生じる可能性があります。

1. ノードリソースの設定 で説明されているように、安全でない sysctl を適切な ノード設定マップ ファイルの **kubeletArguments** パラメーターに追加します。
以下に例を示します。

```
$ oc edit cm node-config-compute -n openshift-node

...
    kubeletArguments:
      allowed-unsafe-sysctls: 1
      - "net.ipv4.tcp_keepalive_time"
      - "net.ipv4.tcp_keepalive_intvl"
      - "net.ipv4.tcp_keepalive_probes"
```

1 使用する安全でない sysctl を追加します。

2. 制限付き SCC の内容を使用し、安全でない sysctl を追加する新規 SCC を作成します。

```
...
allowHostDirVolumePlugin: false
allowHostIPC: false
allowHostNetwork: false
allowHostPID: false
allowHostPorts: false
allowPrivilegeEscalation: true
allowPrivilegedContainer: false
allowedCapabilities: null
allowedUnsafeSysctls: 1
- net.ipv4.tcp_keepalive_time
- net.ipv4.tcp_keepalive_intvl
- net.ipv4.tcp_keepalive_probes
...
metadata:
  name: restricted-sysctls 2
...
```

1 使用する安全でない sysctl を追加します。

2 CR の名前を指定します。

3. 新規 SCC アクセスを Pod ServiceAccount に付与します。

```
$ oc adm policy add-scc-to-user restricted-sysctls -z default -n your_project_name
```

4. 安全でない sysctl を Pod の DeploymentConfig に追加します。

```
kind: DeploymentConfig

...
  template:
    ...
    spec:
      containers:
      ...
      securityContext:
        sysctls:
        - name: net.ipv4.tcp_keepalive_time
          value: "300"
        - name: net.ipv4.tcp_keepalive_intvl
          value: "20"
        - name: net.ipv4.tcp_keepalive_probes
          value: "3"
```

5. 変更を適用するためにノードサービスを再起動します。

```
# systemctl restart atomic-openshift-node
```

33.6. POD の SYSCTL 設定

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

以下の例では、Pod **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。

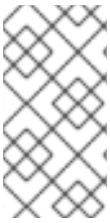
警告

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。

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"
...
```



注記

上記の安全でない sysctl が指定された Pod は、管理者がそれらの 2 つの安全でない sysctl を明示的に有効にしていないノードでの起動に失敗します。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) 機能 または ノードのラベル を使用します。

第34章 データストア層でのデータの暗号化

34.1. 概要

このトピックでは、データストア層でシークレットデータの暗号化を有効にし、これを設定する方法について説明します。サンプルでは **secrets** リソースを使用していますが、**configmaps** などのすべてのリソースを暗号化することができます。

重要

この機能を使用するには、etcd v3 以降が必要になります。

34.2. 設定および暗号がすでに有効にされているかどうかの判別

データ暗号化をアクティブにするには、**--experimental-encryption-provider-config** 引数を Kubernetes API サーバーに渡します。

master-config.yaml の抜粋

```
kubernetesMasterConfig:
  apiServerArguments:
    experimental-encryption-provider-config:
    - /path/to/encryption-config.yaml
```

master-config.yaml およびその形式についての詳細は、マスター設定ファイル のトピックを参照してください。

34.3. 暗号化設定について

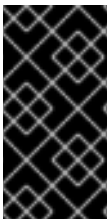
すべての利用可能なプロバイダーを含む暗号化設定ファイル

```
kind: EncryptionConfig
apiVersion: v1
resources: (1)
  - resources: (2)
    - secrets
    providers: (3)
    - aescbc: (4)
        keys:
        - name: key1 (5)
          secret: c2VjcmV0IGlzIHNlY3VyZQ== (6)
        - name: key2
          secret: dGhpcyBpcyBwYXNzd29yZA==
    - secretbox:
        keys:
        - name: key1
          secret: YWJjZGVmZ2hpamtsbW5vcHFyc3R1dnd4eXoxMjM0NTY=
    - aesgcm:
        keys:
        - name: key1
          secret: c2VjcmV0IGlzIHNlY3VyZQ==
```

```
      - name: key2
        secret: dGhpcyBpcyBwYXNzd29yZA==
    - identity: {}
```

1. **resources** のそれぞれの配列項目は分離した設定であり、詳細な設定が含まれます。
2. **resources.resources** フィールドは暗号化が必要な Kubernetes リソース名 (**resource** または **resource.group**) の配列です。
3. **providers** 配列は、順序付けられた 使用可能な暗号化プロバイダーの一覧 です。エントリーごとに 1 つのプロバイダータイプ (**identity** または **aescbc**) のみを指定できますが、同じ項目に両方を指定することはできません。
4. 一覧の最初のプロバイダーがストレージに移動するリソースを暗号化するために使用されます。
5. シークレットの任意の名前です。
6. Base64 のエンコーディングされたランダムキーです。異なるプロバイダーが異なるキーの長さを指定します。詳細は、キーの生成方法 の説明を参照してください。

ストレージからのリソースの読み取り時に、保存されたデータに一致する各プロバイダーはデータの復号化を順番に試行します。情報またはシークレットキーの不一致により保存データを読み取れるプロバイダーがない場合にエラーが返され、クライアントがそのリソースにアクセスできなくなります。



重要

リソースが暗号化設定で読み取れない場合 (キーの変更により)、キーを基礎となる etcd ディレクトリーから削除することのみが必要になります。そのリソースの読み取りを試行する呼び出しは、キーが削除されるか、または有効な復号化キーが提供されない限り失敗します。

34.3.1. 利用可能なプロバイダー

名前	暗号化	強度	速度	キーの長さ	他の考慮事項
identity	なし	該当なし	該当なし	該当なし	暗号化なしのそのままの状態で作成されたリソースです。最初のプロバイダーとして設定される場合、リソースは新規の値が書き込まれるときに復号化されます。
aescbc	PKCS#7 パディングが設定された AES-CBC	最も強い	高速	32 バイト	暗号化に推奨されるオプションですが、**secretbox** よりも若干遅くなる可能性があります。
secretbox	XSalsa20 および Poly1305	強い	より高速	32 バイト	より新しい標準であり、高レベルのレビューが必要な環境では受け入れ可能と見なされない可能性があります。

名前	暗号化	強度	速度	キーの長さ	他の考慮事項
aesgcm	AES-GCM およびランダム初期化ベクター (IV)	200,000 回の書き込みごとにローテーションが必要です。	最速	16、24、または 32 バイト	自動化されたキーの回転スキームが実行される場合以外には、**使用することが推奨されません**。

各プロバイダーは複数のキーをサポートします。キーは復号化の順序で試行されます。プロバイダーが最初のプロバイダーの場合、最初のキーが暗号化に使用されます。

注記

Kubernetes には適切な nonce ジェネレーターがないため、AES-GCM の nonce としてランダム IV を使用します。AES-GCM では適切な nonce がセキュアな状態であることが求められるため、AES-GCM は推奨されません。200,000 回の書き込み制限は nonce の致命的な誤用の可能性を比較的低く抑えます。

34.4. データの暗号化

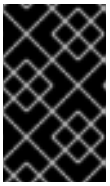
新規の暗号化設定ファイルを作成します。

```
kind: EncryptionConfig
apiVersion: v1
resources:
  - resources:
    - secrets
    providers:
    - aescbc:
        keys:
        - name: key1
          secret: <BASE 64 ENCODED SECRET>
    - identity: {}
```

新規シークレットを作成するには、以下を実行します。

1. 32 バイトのランダムキーを生成し、これを base64 でエンコーディングします。たとえば、Linux および macOS では、以下を使用します。

```
$ head -c 32 /dev/urandom | base64
```



重要

暗号化キーは、**/dev/urandom** などの暗号で保護された適切な乱数ジェネレーターで生成する必要があります。Golang の **math/random** や Python の **random.random()** などは適していません。

2. この値を **secret** フィールドに配置します。

3. API サーバーを再起動します。

```
# master-restart api
# master-restart controllers
```

重要

暗号化プロバイダー設定ファイルには、etcd の内容を復号化できるキーが含まれるため、マスター API サーバーを実行するユーザーのみがこれを読み取れるようにマスターでパーミッションを適切に制限する必要があります。

34.5. データが暗号化されていることの確認

データは etcd に書き込まれる際に暗号化されます。API サーバーの再起動後、新たに作成されたか、または更新されたシークレットは、保存時に暗号化する必要があります。これを確認するには、**etcdctl** コマンドラインプログラムを使用して、シークレットの内容を検索できます。

1. **default** の namespace に、 **secret1** という新規シークレットを作成します。

```
$ oc create secret generic secret1 -n default --from-literal=mykey=mydata
```

2. **etcdctl** コマンドラインを使用し、etcd からシークレットを読み取ります。

```
$ ETCDCTL_API=3 etcdctl get /kubernetes.io/secrets/default/secret1 -w fields [...] | grep
Value
```

[...] には、etcd サーバーに接続するために追加の引数を指定する必要があります。

最終的なコマンドは以下と同様になります。

```
$ ETCDCTL_API=3 etcdctl get /kubernetes.io/secrets/default/secret1 -w fields \
--cacert=/var/lib/origin/openshift.local.config/master/ca.crt \
--key=/var/lib/origin/openshift.local.config/master/master.etcd-client.key \
--cert=/var/lib/origin/openshift.local.config/master/master.etcd-client.crt \
--endpoints 'https://127.0.0.1:4001' | grep Value
```

3. 上記のコマンド出力には、**aescbc** プロバイダーが結果として生成されるデータを暗号化したことを示す **k8s:enc:aescbc:v1:** の接頭辞が付けられます。

4. シークレットが API 経由で取得される場合は、正しく復号化されていることを確認します。

```
$ oc get secret secret1 -n default -o yaml | grep mykey
```

これは **mykey: bXlkYXRh** と一致するはずです。

34.6. すべてのシークレットが暗号化されていることの確認

シークレットは書き込み時に暗号化されるため、シークレットの更新を実行するとその内容は暗号化されることになります。

```
$ oc adm migrate storage --include=secrets --confirm
```

このコマンドはすべてのシークレットを読み取り、次にサーバー側の暗号化を適用するようにそれらを更新します。書き込みの競合のためにエラーが発生する場合は、コマンドを再試行してください。

大規模クラスターの場合、シークレットを namespace 別に細分化するか、または更新をスクリプト化することができます。

34.7. 復号化キーのローテーション

複数の API サーバーが実行されている高可用デプロイメントがある場合などに、ダウンタイムを発生させずにシークレットを変更するには、複数の手順からなる操作が必要になります。

1. 新規キーを生成し、これをすべてのサーバーで同時プロバイダーの 2 つ目のキーエントリーとして追加します。
2. すべての API サーバーを再起動し、各サーバーが新規キーを使用して復号化できるようにします。

注記

単一 API サーバーを使用している場合は、この手順を省略できます。

```
# master-restart api
# master-restart controllers
```

3. 新規キーを **keys** 配列の最初のエントリーにし、これが設定で暗号化に使用されるようにします。
4. すべての API サーバーを再起動し、各サーバーが新規キーを使用して暗号化できるようにします。

```
# master-restart api
# master-restart controllers
```

5. 以下を実行し、新規キーですべての既存シークレットを暗号化します。

```
$ oc adm migrate storage --include=secrets --confirm
```

6. 新規キーを使用して etcd をバックアップし、すべてのシークレットを更新した後に、古い復号化キーを設定から削除します。

34.8. データの復号化

データストア層で暗号化を無効にするには、以下を実行します。

1. **identity** プロバイダーを、設定の最初のエントリーとして配置します。

```
kind: EncryptionConfig
apiVersion: v1
resources:
  - resources:
    - secrets
    providers:
```

```
- identity: {}
- aescbc:
    keys:
    - name: key1
      secret: <BASE 64 ENCODED SECRET>
```

1. すべての API サーバーを再起動します。

```
# master-restart api
# master-restart controllers
```

2. 以下を実行し、すべてのシークレットの復号化を強制的に実行します。

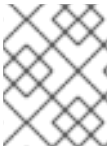
```
$ oc adm migrate storage --include=secrets --confirm
```

第35章 IPSEC を使用したノード間のトラフィックの暗号化

35.1. 概要

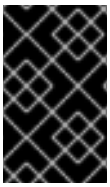
IPsec は、インターネットプロトコル (IP) を使用して通信するすべてのマスターとノードホスト間の通信を暗号化することによって OpenShift Container Platform クラスターのトラフィックを保護します。

このトピックでは、すべてのクラスター管理および Pod データトラフィックを含め、OpenShift Container Platform ホストが IP アドレスを受信する IP サブセット全体の通信のセキュリティーを保護する方法について説明します。



注記

OpenShift Container Platform 管理トラフィックは HTTPS を使用するため、IPsec の有効化により 2 度目の管理トラフィックの暗号化が実行されることになります。



重要

この手順はクラスターの各マスターホストで繰り返し、その後にノードホストで実行される必要があります。IPsec が有効にされていないホストは、これが有効にされているホストと通信することができません。

35.2. ホストの暗号化

前提条件

- **libreswan** 3.15 以降がクラスターホストにインストールされていることを確認します。便宜的なグループ (opportunistic group) 機能が必要な場合は、**libreswan** バージョン 3.19 以降が必要になります。
- このトピックでは、**62** バイトを必要とする IPSec 設定について説明します。クラスターが最大伝送単位 (MTU) 値 **1500** バイトを持つイーサネットネットワーク上で動作している場合、IPsec および SDN のカプセル化のオーバーヘッドに対応するために SDN MTU 値は **1388** バイトに変更する必要があります。クラスター内のノードの MTU を変更するには、以下の手順を実施します。
  a. **node-config-master**、**node-config-infra**、**node-config-compute** の各 ConfigMap を編集します。
     i. 以下のコマンドを実行して ConfigMap を編集します。**<config_map>** を、編集する ConfigMap の名前に置き換えます。

```
# oc edit cm <config_map> -n openshift-node
```

     ii. **mtu** パラメーターを、IPsec オーバーヘッドに十分な MTU サイズ (**1388** バイト) に更新します。

```
networkConfig:
  mtu: 1388
```

  b. 以下のコマンドを実行して SDN インターフェイスを削除します。**<ovs_pod_name>** を OVS Pod の名前に置き換えます。

```
# oc exec <ovs_pod_name> -- ovs-vsctl del-br br0
```

c. クラスターの各ノードについて、以下のアクションを実行します。

i. 更新された MTU 値が **/etc/origin/node/node-config.yaml** ファイルに保存されていることを確認します。

ii. 以下のコマンドを実行して SDN および OVS Pod を再起動します。

```
# oc delete pod -n openshift-sdn -l=app=ovs
# oc delete pod -n openshift-sdn -l=app=sdn
```

35.2.1. 証明書での IPsec の設定

OpenShift Container Platform の内部認証局 (CA) を使用して、IPsec に適した RSA 鍵を生成できます。内部 CA の共通名 (CN) の値は **openshift-signer** に設定されています。

1. 以下のコマンドを実行して、マスターノードで RSA 証明書を生成します。

```
# export CA=/etc/origin/master

# oc adm ca create-server-cert \
  --signer-cert=$CA/ca.crt --signer-key=$CA/ca.key \
  --signer-serial=$CA/ca.serial.txt \
  --hostnames='<hostname>' \ 1
  --cert=<hostname>.crt \ 2
  --key=<hostname>.key 3
```

1 2 3 **<hostname>** をノードの完全修飾ドメイン名 (FQDN) に置き換えます。

2. **openssl** を使用してクライアント証明書、CA 証明書、およびプライベートキーファイルを **PKCS#12** ファイルに追加します。 これは、複数の証明書およびキーの共通ファイル形式です。

```
# openssl pkcs12 -export \
  -in <hostname>.crt \ 1
  -inkey <hostname>.key \ 2
  -certfile  /etc/origin/master/ca.crt \
  -passout pass: \
  -out <hostname>.p12 3
```

1 2 3 ノードの完全修飾ドメイン名 (FQDN) で置き換えます。

3. クラスターの各ノードで、前の手順でホスト用に作成したファイルを安全に転送してから、**PKCS#12** ファイルを **libreswan** 証明書データベースにインポートします。
-W オプションは、パスワードが一時的で **PKCS#12** ファイルに割り当てられないため、空のままになります。

```
# ipsec initnss
# pk12util -i <hostname>.p12 -d sql:/etc/ipsec.d -W ""
# rm <hostname>.p12
```

35.2.2. libreswan IPsec ポリシーの作成

必要な証明書が **libreswan** 証明書データベースにインポートされた後に、それらを使用してクラスター内のホスト間の通信をセキュリティー保護するポリシーを作成します。

libreswan 3.19 以降を使用している場合、便宜的なグループ (opportunistic group) 設定が推奨されます。これ以外の場合は、明示的な接続が必要になります。

35.2.2.1. 便宜的なグループ (opportunistic group) の設定

以下の設定は 2 つの **libreswan** 接続を作成します。最初の設定は OpenShift Container Platform 証明書を使用してトラフィックを暗号化し、2 つ目の設定はクラスターの外部トラフィック用に暗号化に対する例外を作成します。

1. 以下を **/etc/ipsec.d/openshift-cluster.conf** ファイルに配置します。

```
conn private
 left=%defaultroute
 leftid=%fromcert
 # our certificate
 leftcert="NSS Certificate DB:<cert_nickname>" (1)
 right=%opportunisticgroup
 rightid=%fromcert
 # their certificate transmitted via IKE
 rightca=%same
 ikev2=insist
 authby=rsasig
 failureshunt=drop
 negotiationshunt=hold
 auto=ondemand
 encapsulation=yes (2)

conn clear
 left=%defaultroute
 right=%group
 authby=never
 type=passthrough
 auto=route
 priority=100
```

(1) <cert_nickname> を、手順 1 の証明書ニックネームに置き換えます。

(2) NAT を使用しない場合は、カプセル化を強制するには、設定に **encapsulation=yes** を含める必要があります。**Amazon** および **Azure** 内部クラウドネットワークは、IPsec **ESP** または **AH** パケットをルーティングしません。これらのパケットは **UDP** でカプセル化する必要があり、設定されている場合は、NAT の検出が **UDP** のカプセル化で **ESP** を設定します。NAT を使用する場合、または前述のネットワーク/クラウドプロバイダーの制限を受けていない場合は、このパラメーターと値を省略してください。

2. **libreswan** に対して、**/etc/ipsec.d/policies/** のポリシーファイルを使用して各ポリシーを適用する IP サブネットおよびホストを示します。このファイルには、それぞれの設定された接続に対応するポリシーファイルがあります。そのため、上記の例では、**private** および **clear** の 2 つの接続のそれぞれに **/etc/ipsec.d/policies/** のファイルが設定されます。
/etc/ipsec.d/policies/private には、ホストの IP アドレスの受信元であるクラスターの IP サブネットが含まれる必要があります。これにより、デフォルトでは、リモートホストのクライ

アント証明書がローカルホストの認証局の証明書に対して認証される場合、クラスターサブネットのホスト間のすべての通信が暗号化されることになります。リモートホストの証明書が認証されない場合、2 つのホスト間のすべてのトラフィックがブロックされます。

たとえば、すべてのホストが **172.16.0.0/16** アドレス空間のアドレスを使用するように設定される場合、**private** ポリシーファイルには **172.16.0.0/16** が含まれることになります。暗号化する追加サブセットの任意の数がこのファイルに追加され、それらのサブネットへのすべてのトラフィックでも IPsec が使用されることになります。

3. トラフィックがクラスターに出入りすることを確認するためにすべてのホストとサブネットゲートウェイ間の通信の暗号化を解除します。ゲートウェイを **/etc/ipsec.d/policies/clear** ファイルに追加します。

```
172.16.0.1/32
```

追加のホストおよびサブネットをこのファイルに追加できます。これにより、これらのホストおよびサブネットへのすべてのトラフィックの暗号が解除されます。

35.2.2.2. 明示的な接続の設定

この設定では、各 IPSec ノード設定がクラスター内の他のすべてのノードの設定を明示的に一覧表示する必要があります。各ノードで Ansible などの設定管理ツールを使用してこのファイルを生成することが推奨されます。

注記

node-config.yaml ファイルは手動で変更しないようにしてください。クラスターのノードを変更するには、ノード設定マップ を必要に応じて更新します。

また、この設定では各ノードの完全な証明書サブジェクトをその他のノードの設定に配置する必要があります。

1. **openssl** を使用して、このサブジェクトをノードの証明書から読み取ります。

```
# openssl x509 \
  -in /path/to/client-certificate -text | \
  grep "Subject:" | \
  sed 's/[[:blank:]]*Subject: //'
```

2. 以下の行を、クラスター内の他のすべてのノードの各ノードの **/etc/ipsec.d/openshift-cluster.conf** ファイルに配置します。

```
conn <other_node_hostname>
        left=<this_node_ip> 1
        leftid="CN=<this_node_cert_nickname>" 2
        leftrsasigkey=%cert
        leftcert=<this_node_cert_nickname> 3
        right=<other_node_ip> 4
        rightid="<other_node_cert_full_subject>" 5
        rightrsasigkey=%cert
        auto=start
        keyingtries=%forever
 encapsulation=yes 6
```

1. <this_node_ip> をこのノードのクラスター IP アドレスに置き換えます。
2. 3. <this_node_cert_nickname> を手順 1 のノードの証明書ニックネームに置き換えます。
4. <other_node_ip> を他のノードのクラスター IP アドレスに置き換えます。
5. <other_node_cert_full_subject> を上記の他のノードの証明書に置き換えます。たとえば、"O=system:nodes,CN=openshift-node-45.example.com" のようになります。
6. NAT を使用しない場合は、カプセル化を強制するには、設定に **encapsulation=yes** を含める必要があります。**Amazon** および **Azure** 内部クラウドネットワークは、IPsec **ESP** または **AH** パケットをルーティングしません。これらのパケットは **UDP** でカプセル化する必要があり、設定されている場合は、NAT の検出が **UDP** のカプセル化で **ESP** を設定します。NAT を使用する場合、または前述のネットワーク/クラウドプロバイダーの制限を受けていない場合は、このパラメーターと値を省略してください。

3. 以下を各ノードの **/etc/ipsec.d/openshift-cluster.secrets** ファイルに配置します。

```
: RSA "<this_node_cert_nickname>" 1
```

1. <this_node_cert_nickname> を手順 1 のノードの証明書ニックネームに置き換えます。

35.3. IPSEC ファイアウォールの設定

クラスター内のすべてのノードは、IPSec 関連のネットワークトラフィックを許可する必要があります。これには、UDP ポート 500 だけでなく IP プロトコル番号の 50 と 51 が含まれます。

たとえば、クラスターノードがインターフェイス **eth0** で通信する場合、以下のようになります。

```
-A OS_FIREWALL_ALLOW -i eth0 -p 50 -j ACCEPT
-A OS_FIREWALL_ALLOW -i eth0 -p 51 -j ACCEPT
-A OS_FIREWALL_ALLOW -i eth0 -p udp --dport 500 -j ACCEPT
```

注記

また IPSec は、NAT traversal に UDP ポート 4500 を使用します。 ただし、これは通常のクラスターデプロイメントに適用することはできません。

35.4. IPSEC の開始および終了

1. **ipsec** サービスを開始し、新規の設定およびポリシーを読み込み、暗号化を開始します。

```
# systemctl start ipsec
```

2. **ipsec** サービスを有効にして起動時に開始します。

```
# systemctl enable ipsec
```

35.5. IPSEC の最適化

IPSec を使用した暗号化の場合のパフォーマンスに関する提案の詳細は、Scaling and Performance Guide を参照してください。

35.6. トラブルシューティング

2 つのノード間で認証を完了できない場合、すべてのトラフィックが拒否されるため、それらの間で ping を行うことはできません。**clear** ポリシーが適切に設定されていない場合も、クラスター内の別のホストから SSH をホストに対して実行することはできません。

ipsec status コマンドを使用して **clear** および **private** ポリシーが読み込まれていることを確認できます。

第36章 依存関係ツリーのビルド

36.1. 概要

OpenShift Container Platform は、image stream tag の更新時を検知するために、**BuildConfig** の image change triggers を使用します。**oc adm build-chain** コマンドを使用して、指定された イメージストリーム でイメージを更新することによって影響を受ける イメージ を特定する依存関係ツリーをビルドできます。

build-chain ツールは、トリガーする ビルド を決定できます。それらのビルドの出力を分析して、別の イメージストリームタグ を更新するかどうかを判断します。その場合、これらのツールは引き続き依存関係ツリーに従います。更新する場合、このツールは依存関係ツリーを引き続きフォローします。最後に、トップレベルのタグの更新によって影響を受けるイメージストリームタグを指定するグラフを出力します。 このツールのデフォルトの出力構文は人間が判読できる形式に設定されます。 DOT 形式もサポートされます。

36.2. 使用法

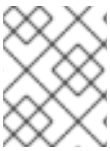
以下の表は、よく使用される **build-chain** の使用方法と一般的な構文について説明しています。

表36.1 よく使用される **build-chain** 操作

説明	構文
<image-stream> の **最新** タグの依存関係ツリーをビルドします。	`$ oc adm build-chain <image-stream>`

DOT 形式で **v2** タグの依存関係ツリーをビルドし、DOT ユーティリティーを使用してこれを可視化します。	`$ oc adm build-chain <image-stream>:v2 \`
	`  -o dot \`
	`  | dot -T svg -o deps.svg`

test プロジェクトにある指定されたイメージストリームタグについての全プロジェクト間の依存関係ツリーをビルドします。	`$ oc adm build-chain <image-stream>:v1 \` `  -n test --all`



注記

graphviz パッケージをインストールして **dot** コマンドを使用する必要がある場合があります。

第37章 失敗した ETCD メンバーの置き換え

一部の etcd メンバーが失敗しても、依然として etcd メンバーのクォーラム (定足数) がある場合、残りの etcd メンバーおよびそれらに含まれるデータを使用して、etcd またはクラスターのダウンタイムなしに etcd メンバーを追加できます。

37.1. 失敗した ETCD メンバーの削除

新規の etcd ノードを追加する前に、失敗したノードを削除します。

手順

1. アクティブな etcd ホストから、失敗した etcd ノードを削除します。

```
# etcdctl -C https://<surviving host IP>:2379 \
  --ca-file=/etc/etcd/ca.crt     \
  --cert-file=/etc/etcd/peer.crt     \
  --key-file=/etc/etcd/peer.key cluster-health

# etcdctl -C https://<surviving host IP>:2379 \
  --ca-file=/etc/etcd/ca.crt     \
  --cert-file=/etc/etcd/peer.crt     \
  --key-file=/etc/etcd/peer.key member remove <failed member identifier>
```

2. etcd Pod 定義を削除して、失敗した etcd メンバーで etcd サービスを停止します。

```
# mkdir -p /etc/origin/node/pods-stopped
# mv /etc/origin/node/pods/* /etc/origin/node/pods-stopped/
```

3. **etcd** ディレクトリーの内容を削除します。

重要

コンテンツを削除する前に、このディレクトリーをクラスター外の場所にバックアップすることが推奨されます。復元が成功した後、このバックアップを削除できます。

```
# rm -rf /var/lib/etcd/*
```

37.2. ETCD メンバーの追加

etcd ホストは、Ansible Playbook または手動の手順のいずれかを使用して追加できます。

37.2.1. Ansible を使用した新規 etcd ホストの追加

手順

1. Ansible インベントリーファイルで、**[new_etcd]** という名前の新規グループおよび新規ホストを作成します。次に、**new_etcd** グループを **[OSEv3]** グループの子として追加します。

```
[OSEv3:children]
masters
```

```
nodes
etcd
new_etcd (1)

... [OUTPUT ABBREVIATED] ...

[etcd]
master-0.example.com
master-1.example.com
master-2.example.com

[new_etcd] (2)
etcd0.example.com (3)
```

(1)(2)(3)これらの行を追加します。

注記

インベントリーファイル内の古い **etcd host** エントリーを新しい **etcd host** エントリーに置き換えます。古い **etcd host** を置き換えるときに、**/etc/etcd/ca/** ディレクトリーのコピーを作成する必要があります。または、**etcd hosts** をスケールアップする前に、etcd ca と証明書を再デプロイすることもできます。

2. OpenShift Container Platform をインストールし、Ansible インベントリーファイルをホストするホストから、Playbook ディレクトリーに移動し、etcd **scaleup** Playbook を実行します。

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook  playbooks/openshift-etcd/scaleup.yml
```

3. Playbook が実行された後に、新規 etcd ホストを **[new_etcd]** グループから **[etcd]** グループに移行し、現在のステータスを反映するようにインベントリーファイルを変更します。

```
[OSEv3:children]
masters
nodes
etcd
new_etcd

... [OUTPUT ABBREVIATED] ...

[etcd]
master-0.example.com
master-1.example.com
master-2.example.com
etcd0.example.com
```

4. Flannel を使用する場合には、OpenShift Container Platform のホストごとに、**/etc/sysconfig/flanneld** にある **flanneld** サービス設定を変更し、新しい etcd ホストを追加します。

```
FLANNEL_ETCD_ENDPOINTS=https://master-0.example.com:2379,https://master-
1.example.com:2379,https://master-2.example.com:2379,https://etcd0.example.com:2379
```

5. **flanneld** サービスを再起動します。

```
# systemctl restart flanneld.service
```

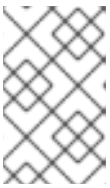
37.2.2. 新規 etcd ホストの手動による追加

etcd をマスターノードで静的 Pod として実行しない場合、別の etcd ホストを追加する必要が生じる場合があります。

手順
現在の **etcd** クラスターの変更
etcd 証明書を作成するには、値を環境の値に置き換えて **openssl** コマンドを実行します。

1. 環境変数を作成します。

```
export NEW_ETCD_HOSTNAME="*etcd0.example.com*"
export NEW_ETCD_IP="192.168.55.21"

export CN=$NEW_ETCD_HOSTNAME
export SAN="IP:${NEW_ETCD_IP}, DNS:${NEW_ETCD_HOSTNAME}"
export PREFIX="/etc/etcd/generated_certs/etcd-$CN/"
export OPENSSLCFG="/etc/etcd/ca/openssl.cnf"
```



注記

etcd_v3_ca_* として使用されるカスタムの **openssl** 拡張には、**subjectAltName** としての $SAN 環境変数が含まれます。詳細は、**/etc/etcd/ca/openssl.cnf** を参照してください。

2. 設定および証明書を保存するディレクトリーを作成します。

```
# mkdir -p ${PREFIX}
```

3. サーバー証明書要求を作成し、これに署名します (**server.csr** および **server.crt**)。

```
# openssl req -new -config ${OPENSSLCFG} \
    -keyout ${PREFIX}server.key  \
    -out ${PREFIX}server.csr \
    -reqexts etcd_v3_req -batch -nodes \
    -subj /CN=$CN

# openssl ca -name etcd_ca -config ${OPENSSLCFG} \
    -out ${PREFIX}server.crt \
    -in ${PREFIX}server.csr \
    -extensions etcd_v3_ca_server -batch
```

4. ピア証明書要求を作成し、これに署名します (**peer.csr** および **peer.crt**)。

```
# openssl req -new -config ${OPENSSLCFG} \
    -keyout ${PREFIX}peer.key \
    -out ${PREFIX}peer.csr \
    -reqexts etcd_v3_req -batch -nodes \
    -subj /CN=$CN
```

```
# openssl ca -name etcd_ca -config ${OPENSSLCFG} \
  -out ${PREFIX}peer.crt \
  -in ${PREFIX}peer.csr \
  -extensions etcd_v3_ca_peer -batch
```

5. 後で変更できるように、現在の etcd 設定および **ca.crt** ファイルをサンプルとして現在のノードからコピーします。

```
# cp /etc/etcd/etcd.conf ${PREFIX}
# cp /etc/etcd/ca.crt ${PREFIX}
```

6. 存続する etcd ホストから、新規ホストをクラスターに追加します。etcd メンバーをクラスターに追加するには、まず最初のメンバーの **peerURLs** 値のデフォルトの **localhost** ピアを調整する必要があります。

   a. **member list** コマンドを使用して最初のメンバーのメンバー ID を取得します。

```
# etcdctl --cert-file=/etc/etcd/peer.crt \
    --key-file=/etc/etcd/peer.key \
    --ca-file=/etc/etcd/ca.crt \
    --peers="https://172.18.1.18:2379,https://172.18.9.202:2379,https://172.18.0.75:2379"
\ (1)
    member list
```

   (1) **--peers** パラメーター値でアクティブな etcd メンバーのみの URL を指定するようにしてください。

   b. etcd がクラスターピアについてリッスンする IP アドレスを取得します。

```
$ ss -l4n | grep 2380
```

   c. 直前の手順で取得されたメンバー ID および IP アドレスを渡して、**etcdctl member update** コマンドを使用して **peerURLs** の値を更新します。

```
# etcdctl --cert-file=/etc/etcd/peer.crt \
    --key-file=/etc/etcd/peer.key \
    --ca-file=/etc/etcd/ca.crt \
    --peers="https://172.18.1.18:2379,https://172.18.9.202:2379,https://172.18.0.75:2379"
\
    member update 511b7fb6cc0001 https://172.18.1.18:2380
```

   d. **member list** コマンドを再実行し、ピア URL に **localhost** が含まれなくなるようにします。

7. 新規ホストを etcd クラスターに追加します。新規ホストはまだ設定されていないため、新規ホストを設定するまでステータスが **unstarted** のままであることに注意してください。

警告

各メンバーを追加し、1回に1つずつメンバーをオンライン状態にします。各メンバーをクラスターに追加する際に、現在のピアの **peerURL** 一覧を調整する必要があります。**peerURL** 一覧はメンバーが追加されるたびに拡張します。**etcdctl member add** コマンドは、以下に説明されているように、メンバーを追加する際に **etcd.conf** ファイルで設定する必要のある値を出力します。

```
# etcdctl -C https://${CURRENT_ETCD_HOST}:2379 \
  --ca-file=/etc/etcd/ca.crt     \
  --cert-file=/etc/etcd/peer.crt     \
  --key-file=/etc/etcd/peer.key member add ${NEW_ETCD_HOSTNAME}
https://${NEW_ETCD_IP}:2380 1

Added member named 10.3.9.222 with ID 4e1db163a21d7651 to cluster

ETCD_NAME="<NEW_ETCD_HOSTNAME>"
ETCD_INITIAL_CLUSTER="<NEW_ETCD_HOSTNAME>=https://<NEW_HOST_IP>:2380,
<CLUSTERMEMBER1_NAME>=https:/<CLUSTERMEMBER2_IP>:2380,
<CLUSTERMEMBER2_NAME>=https:/<CLUSTERMEMBER2_IP>:2380,
<CLUSTERMEMBER3_NAME>=https:/<CLUSTERMEMBER3_IP>:2380"
ETCD_INITIAL_CLUSTER_STATE="existing"
```

1 この行で、**10.3.9.222** は etcd メンバーのラベルです。ホスト名、IP アドレスまたは単純な名前を指定できます。

8. サンプル **${PREFIX}/etcd.conf** ファイルを更新します。

   a. 以下の値を直前の手順で生成された値に置き換えます。

      - ETCD_NAME
      - ETCD_INITIAL_CLUSTER
      - ETCD_INITIAL_CLUSTER_STATE

   b. 以下の変数は、直前の手順で出力された新規ホストの IP に変更します。**${NEW_ETCD_IP}** は、値として使用できます。

      ```
      ETCD_LISTEN_PEER_URLS
      ETCD_LISTEN_CLIENT_URLS
      ETCD_INITIAL_ADVERTISE_PEER_URLS
      ETCD_ADVERTISE_CLIENT_URLS
      ```

   c. メンバーシステムを etcd ノードとして使用していた場合には、**/etc/etcd/etcd.conf** ファイルの現在の値を上書きする必要があります。

   d. ファイルで構文エラーや欠落している IP アドレスがないかを確認します。 エラーや欠落がある場合には、etced サービスが失敗してしまう可能性があります。

```
# vi ${PREFIX}/etcd.conf
```

9. インストールファイルをホストするノードでは、**/etc/ansible/hosts** インベントリーファイルの **[etcd]** ホストグループを更新します。古い etcd ホストを削除し、新規ホストを追加します。

10. 証明書、サンプル設定ファイル、および **ca** を含む **tgz** ファイルを作成し、これを新規ホストにコピーします。

```
# tar -czvf /etc/etcd/generated_certs/${CN}.tgz -C ${PREFIX} .
# scp /etc/etcd/generated_certs/${CN}.tgz ${CN}:/tmp/
```

新規 **etcd** ホストの変更

1. **iptables-services** をインストールして etcd の必要なポートを開くために iptables ユーティリティーを指定します。

```
# yum install -y iptables-services
```

2. etcd の通信を許可する **OS_FIREWALL_ALLOW** ファイアウォールルールを作成します。
   - クライアントのポート 2379/tcp
   - ピア通信のポート 2380/tcp

```
# systemctl enable iptables.service --now
# iptables -N OS_FIREWALL_ALLOW
# iptables -t filter -I INPUT -j OS_FIREWALL_ALLOW
# iptables -A OS_FIREWALL_ALLOW -p tcp -m state --state NEW -m tcp --dport 2379 -j ACCEPT
# iptables -A OS_FIREWALL_ALLOW -p tcp -m state --state NEW -m tcp --dport 2380 -j ACCEPT
# iptables-save | tee /etc/sysconfig/iptables
```

注記

この例では、新規チェーン **OS_FIREWALL_ALLOW** が作成されます。 これは、OpenShift Container Platform インストーラーがファイアウォールルールに使用する標準の名前になります。

警告

環境が IaaS 環境でホストされている場合には、インスタンスがこれらのポートに入ってくるトラフィックを許可できるように、セキュリティーグループを変更します。

3. etcd をインストールします。

```
# yum install -y etcd
```

バージョン **etcd-2.3.7-4.el7.x86_64** 以降がインストールされていることを確認します。

4. etcd Pod 定義を削除して、etcd サービスが実行されていない状態にします。

```
# mkdir -p /etc/origin/node/pods-stopped
# mv /etc/origin/node/pods/* /etc/origin/node/pods-stopped/
```

5. etcd 設定およびデータを削除します。

```
# rm -Rf /etc/etcd/*
# rm -Rf /var/lib/etcd/*
```

6. 証明書および設定ファイルを展開します。

```
# tar xzvf /tmp/etcd0.example.com.tgz -C /etc/etcd/
```

7. 新規ホストで etcd を起動します。

```
# systemctl enable etcd --now
```

8. ホストがクラスターの一部であることと現在のクラスターの正常性を確認します。

- v2 etcd api を使用する場合は、以下のコマンドを実行します。

```
# etcdctl --cert-file=/etc/etcd/peer.crt \
        --key-file=/etc/etcd/peer.key \
        --ca-file=/etc/etcd/ca.crt \
        --peers="https://*master-0.example.com*:2379,\
        https://*master-1.example.com*:2379,\
        https://*master-2.example.com*:2379,\
        https://*etcd0.example.com*:2379"\
        cluster-health
member 5ee217d19001 is healthy: got healthy result from https://192.168.55.12:2379
member 2a529ba1840722c0 is healthy: got healthy result from https://192.168.55.8:2379
member 8b8904727bf526a5 is healthy: got healthy result from
https://192.168.55.21:2379
member ed4f0efd277d7599 is healthy: got healthy result from https://192.168.55.13:2379
cluster is healthy
```

- v3 etcd api を使用する場合は、以下のコマンドを実行します。

```
# ETCDCTL_API=3 etcdctl --cert="/etc/etcd/peer.crt" \
        --key=/etc/etcd/peer.key \
        --cacert="/etc/etcd/ca.crt" \
        --endpoints="https://*master-0.example.com*:2379,\
          https://*master-1.example.com*:2379,\
          https://*master-2.example.com*:2379,\
          https://*etcd0.example.com*:2379"\
          endpoint health
https://master-0.example.com:2379 is healthy: successfully committed proposal: took =
5.011358ms
https://master-1.example.com:2379 is healthy: successfully committed proposal: took =
1.305173ms
https://master-2.example.com:2379 is healthy: successfully committed proposal: took =
```

```
1.388772ms
https://etcd0.example.com:2379 is healthy: successfully committed proposal: took =
1.498829ms
```

各 OpenShift Container Platform マスターの変更

1. すべてのマスターの **/etc/origin/master/master-config.yaml** ファイルの **etcClientInfo** セクションでマスター設定を変更します。新規 etcd ホストを、データを保存するために OpenShift Container Platform が使用する etcd サーバーの一覧に追加し、失敗したすべての etcd ホストを削除します。

```
etcdClientInfo:
  ca: master.etcd-ca.crt
  certFile: master.etcd-client.crt
  keyFile: master.etcd-client.key
  urls:
    - https://master-0.example.com:2379
    - https://master-1.example.com:2379
    - https://master-2.example.com:2379
    - https://etcd0.example.com:2379
```

2. マスター API サービスを再起動します。

   - すべてのマスターのインストールに対しては、以下を実行します。

```
# master-restart api
# master-restart controllers
```

警告

etcd ノードの数は奇数でなければなりません。 そのため、2 つ以上のホストを追加する必要があります。

3. Flannel を使用する場合、新規 etcd ホストを組み込むために、すべての OpenShift Container Platform ホストの **/etc/sysconfig/flanneld** にある **flanneld** サービス設定を変更します。

```
FLANNEL_ETCD_ENDPOINTS=https://master-0.example.com:2379,https://master-
1.example.com:2379,https://master-2.example.com:2379,https://etcd0.example.com:2379
```

4. **flanneld** サービスを再起動します。

```
# systemctl restart flanneld.service
```

第38章 ETCD のクォーラム (定足数) の復元

etcd のクォーラム (定足数) を失う場合、それを復元できます。

- etcd を別のホストで実行する場合、etcd をバックアップし、etcd クラスターを停止してからこれを新たに作成する必要があります。1 つの正常な etcd ノードを使用して新規クラスターを作成することができますが、他のすべての正常なノードを削除する必要があります。
- etcd をマスターノード上で静的 Pod として実行する場合、etcd Pod を停止し、一時的なクラスターを作成してから etcd Pod を再起動します。

注記

etcd のクォーラム (定足数) が失われる際に、OpenShift Container Platform で実行されるアプリケーションは影響を受けません。ただし、プラットフォームの機能は読み取り専用の操作に制限されます。アプリケーションの拡大または縮小、デプロイメントの変更、またはビルドの実行または変更などの操作を実行することはできません。

etcd のクォーラム (定足数) が失われていることを確認するには、以下のコマンドのいずれかを実行し、クラスターが正常な状態にないことを確認します。

- etcd v2 API を使用する場合、以下のコマンドを実行します。

```
# etcd_ctl=2 etcdctl  --cert-file=/etc/origin/master/master.etcd-client.crt  \
        --key-file /etc/origin/master/master.etcd-client.key \
        --ca-file /etc/origin/master/master.etcd-ca.crt \
        --endpoints="https://*master-0.example.com*:2379,\
        https://*master-1.example.com*:2379,\
        https://*master-2.example.com*:2379"\
        cluster-health

member 165201190bf7f217 is unhealthy: got unhealthy result from https://master-
0.example.com:2379
member b50b8a0acab2fa71 is unreachable: [https://master-1.example.com:2379] are all
unreachable
member d40307cbca7bc2df is unreachable: [https://master-2.example.com:2379] are all
unreachable
cluster is unhealthy
```

- v3 API を使用する場合、以下のコマンドを実行します。

```
# ETCDCTL_API=3 etcdctl --cert=/etc/origin/master/master.etcd-client.crt  \
        --key=/etc/origin/master/master.etcd-client.key \
        --cacert=/etc/origin/masterca.crt \
        --endpoints="https://*master-0.example.com*:2379,\
        https://*master-1.example.com*:2379,\
        https://*master-2.example.com*:2379"\
        endpoint health
https://master-0.example.com:2379 is unhealthy: failed to connect: context deadline
exceeded
https://master-1.example.com:2379 is unhealthy: failed to connect: context deadline
exceeded
https://master-2.example.com:2379 is unhealthy: failed to connect: context deadline
exceeded
Error:  unhealthy cluster
```

ホストのメンバー ID およびホスト名を書き留めます。到達できるノードのいずれかを使用して新規クラスターを作成します。

38.1. 複数サービスの ETCD クォーラム (定足数) の復元

38.1.1. etcd のバックアップ

etcd のバックアップ時に、etcd 設定ファイルと etcd データの両方をバックアップする必要があります。

38.1.1.1. etcd 設定ファイルのバックアップ

保持する etcd 設定ファイルはすべて etcd が実行されているインスタンスの **/etc/etcd** ディレクトリーに保存されます。これには、etcd 設定ファイル (**/etc/etcd/etcd.conf**) およびクラスターの通信に必要な証明書が含まれます。それらすべてのファイルは Ansible インストーラーによってインストール時に生成されます。

手順
クラスターの各 etcd メンバーについての etcd 設定をバックアップします。

```
$ ssh master-0 1
# mkdir -p /backup/etcd-config-$(date +%Y%m%d)/
# cp -R /etc/etcd/ /backup/etcd-config-$(date +%Y%m%d)/
```

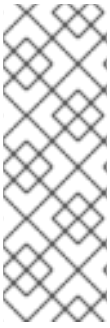
1 **master-0** は、etcd メンバーの名前に置き換えます。

注記

各 etcd クラスターメンバーの証明書および設定ファイルは一意のものです。

38.1.1.2. etcd データのバックアップ

前提条件



注記

OpenShift Container Platform インストーラーはエイリアスを作成するため、**etcdctl2** (etcd v2 タスクの場合) と **etcdctl3** (etcd v3 タスクの場合) という名前のすべてのフラグを入力しなくて済みます。

ただし、**etcdctl3** エイリアスは **etcdctl** コマンドに詳細なエンドポイント一覧を提供しないため、**--endpoints** オプションを指定し、すべてのエンドポイントを一覧表示する必要があります。

etcd をバックアップする前に、以下を確認してください。

- **etcdctl** バイナリーが利用可能であるか、またはコンテナー化インストールの場合は **rhel7/etcd** コンテナーが利用可能でなければなりません。
- OpenShift Container Platform API サービスが実行中であることを確認します。

- etcd クラスターとの接続を確認します (ポート 2379/tcp)。
- etcd クラスターに接続するために使用する適切な証明書があることを確認します。

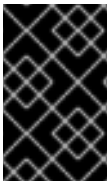
手順

注記

etcdctl backup コマンドはバックアップを実行するために使用されますが、etcd v3 には バックアップ の概念がありません。代わりに、**etcdctl snapshot save** コマンドを使用してライブメンバーの **snapshot** を取るか、または etcd データディレクトリーの **member/snap/db** ファイルをコピーしてください。

etcdctl backup コマンドは、ノード ID やクラスター ID などのバックアップに含まれるメタデータの一部を書き換えるので、バックアップでは、ノードの以前のアイデンティティーが失われます。バックアップからクラスターを再作成するには、新規の単一ノードクラスターを作成してから、残りのノードをクラスターに追加します。メタデータは新規ノードが既存クラスターに加わらないように再作成されます。

etcd データをバックアップします。



重要

OpenShift Container Platform の以前のバージョンからアップグレードしたクラスターには、v2 データストアが含まれる可能性があります。すべての etcd データストアをバックアップしてください。

1. 静的 Pod マニフェストから etcd エンドポイント IP アドレスを取得します。

```
$ export ETCD_POD_MANIFEST="/etc/origin/node/pods/etcd.yaml"
```

```
$ export ETCD_EP=$(grep https ${ETCD_POD_MANIFEST} | cut -d '/' -f3)
```

2. 管理者としてログインします。

```
$ oc login -u system:admin
```

3. etcd Pod 名を取得します。

```
$ export ETCD_POD=$(oc get pods -n kube-system | grep -o -m 1 '^master-etcd\S*')
```

4. **kube-system** プロジェクトに切り替えます。

```
$ oc project kube-system
```

5. Pod の etcd データのスナップショットを作成し、これをローカルに保存します。

```
$ oc exec ${ETCD_POD} -c etcd -- /bin/bash -c "ETCDCTL_API=3 etcdctl \
    --cert /etc/etcd/peer.crt \
    --key /etc/etcd/peer.key \
```

```
    --cacert /etc/etcd/ca.crt \
    --endpoints $ETCD_EP \
    snapshot save /var/lib/etcd/snapshot.db" (1)
```

(1) スナップショットは、**/var/lib/etcd/** 配下のディレクトリーに記述する必要があります。

38.1.2. etcd ホストの削除

復元後に etcd ホストが失敗する場合は、クラスターから削除します。etcd のクォーラム (定足数) が失われた状態から回復するには、クラスターから 1 つの etcd ノード以外のすべての正常な etcd ノードを削除する必要もあります。

すべてのマスターホストで実行する手順

手順

1. etcd クラスターから他の etcd ホストをそれぞれ削除します。それぞれの etcd ノードについて以下のコマンドを実行します。

```
# etcdctl3 --endpoints=https://<surviving host IP>:2379
  --cacert=/etc/etcd/ca.crt
  --cert=/etc/etcd/peer.crt
  --key=/etc/etcd/peer.key member remove <failed member ID>
```

2. すべてのマスターの **/etc/origin/master/master-config.yaml** +master 設定ファイルから他の etcd ホストを削除します。

```
etcdClientInfo:
  ca: master.etcd-ca.crt
  certFile: master.etcd-client.crt
  keyFile: master.etcd-client.key
  urls:
    - https://master-0.example.com:2379
    - https://master-1.example.com:2379 (1)
    - https://master-2.example.com:2379 (2)
```

(1) (2) 削除するホストです。

3. すべてのマスターでマスター API サービスを再起動します。

```
# master-restart api restart-master controller
```

現在の **etcd** クラスターで実行する手順

手順

1. 失敗したホストをクラスターから削除します。

```
# etcdctl2 cluster-health
member 5ee217d19001 is healthy: got healthy result from https://192.168.55.12:2379
member 2a529ba1840722c0 is healthy: got healthy result from https://192.168.55.8:2379
failed to check the health of member 8372784203e11288 on https://192.168.55.21:2379: Get
https://192.168.55.21:2379/health: dial tcp 192.168.55.21:2379: getsockopt: connection
```

```
refused
member 8372784203e11288 is unreachable: [https://192.168.55.21:2379] are all
unreachable
member ed4f0efd277d7599 is healthy: got healthy result from https://192.168.55.13:2379
cluster is healthy

# etcdctl2 member remove 8372784203e11288 1
Removed member 8372784203e11288 from cluster

# etcdctl2 cluster-health
member 5ee217d19001 is healthy: got healthy result from https://192.168.55.12:2379
member 2a529ba1840722c0 is healthy: got healthy result from https://192.168.55.8:2379
member ed4f0efd277d7599 is healthy: got healthy result from https://192.168.55.13:2379
cluster is healthy
```

1 **remove** コマンドにはホスト名ではなく、etcd ID が必要です。

2. etcd 設定で etcd サービスの再起動時に失敗したホストを使用しないようにするには、残りのすべての etcd ホストで **/etc/etcd/etcd.conf** ファイルを変更し、**ETCD_INITIAL_CLUSTER** 変数の値から失敗したホストを削除します。

```
# vi /etc/etcd/etcd.conf
```

以下に例を示します。

```
ETCD_INITIAL_CLUSTER=master-0.example.com=https://192.168.55.8:2380,master-
1.example.com=https://192.168.55.12:2380,master-
2.example.com=https://192.168.55.13:2380
```

以下のようになります。

```
ETCD_INITIAL_CLUSTER=master-0.example.com=https://192.168.55.8:2380,master-
1.example.com=https://192.168.55.12:2380
```

注記

失敗したホストは **etcdctl** を使用して削除されているので、etcd サービスの再起動は不要です。

3. Ansible インベントリーファイルをクラスターの現在のステータスを反映し、Playbook の再実行時の問題を防げるように変更します。

```
[OSEv3:children]
masters
nodes
etcd

... [OUTPUT ABBREVIATED] ...

[etcd]
master-0.example.com
master-1.example.com
```

4. Flannel を使用している場合、すべてのホストの **/etc/sysconfig/flanneld** にある **flanneld** サービス設定を変更し、etcd ホストを削除します。

```
FLANNEL_ETCD_ENDPOINTS=https://master-0.example.com:2379,https://master-1.example.com:2379,https://master-2.example.com:2379
```

5. **flanneld** サービスを再起動します。

```
# systemctl restart flanneld.service
```

38.1.3. 単一ノード etcd クラスターの作成

OpenShift Container Platform インスタンスの完全な機能を復元するには、残りの etcd ノードをスタンドアロン etcd クラスターにします。

手順

1. クラスターから削除しなかった etcd ノードで、etcd の Pod 定義を削除してすべての etcd サービスを停止します。

```
# mkdir -p /etc/origin/node/pods-stopped
# mv /etc/origin/node/pods/etcd.yaml /etc/origin/node/pods-stopped/
# systemctl stop atomic-openshift-node
# mv /etc/origin/node/pods-stopped/etcd.yaml /etc/origin/node/pods/
```

2. ホストで etcd サービスを実行し、新規クラスターを強制的に実行します。これらのコマンドは、**--force-new-cluster** オプションを etcd 起動コマンドに追加する etcd サービスのカスタムファイルを作成します。

```
# mkdir -p /etc/systemd/system/etcd.service.d/
# echo "[Service]" > /etc/systemd/system/etcd.service.d/temp.conf
# echo "ExecStart=" >> /etc/systemd/system/etcd.service.d/temp.conf
# sed -n '/ExecStart/s/"$/ --force-new-cluster"/p' \
    /usr/lib/systemd/system/etcd.service \
    >> /etc/systemd/system/etcd.service.d/temp.conf

# systemctl daemon-reload
# master-restart etcd
```

3. etcd メンバーを一覧表示し、メンバー一覧に単一の etcd ホストのみが含まれることを確認します。

```
# etcdctl member list
165201190bf7f217: name=192.168.34.20 peerURLs=http://localhost:2380 clientURLs=https://master-0.example.com:2379 isLeader=true
```

4. データの復元および新規クラスターの作成後に、**peerURLs** パラメーターを、etcd がピア通信をリッスンする IP アドレスを使用するように更新する必要があります。

```
# etcdctl member update 165201190bf7f217 https://192.168.34.20:2380 1
```

1 **165201190bf7f217** は直前のコマンドの出力に示されるメンバー ID であり、**https://192.168.34.20:2380** はその IP アドレスです。

5. 検証するには、IP がメンバーの一覧にあることを確認します。

```
$ etcdctl2 member list
5ee217d17301: name=master-0.example.com peerURLs=https://*192.168.55.8*:2380
clientURLs=https://192.168.55.8:2379 isLeader=true
```

38.1.4. 復元後の etcd ノードの追加

最初のインスタンスを実行後に、複数の etcd サーバーをクラスターに追加できます。

手順

1. **ETCD_NAME** 変数でインスタンスの etcd 名を取得します。

```
# grep ETCD_NAME /etc/etcd/etcd.conf
```

2. etcd がピア通信をリッスンする IP アドレスを取得します。

```
# grep ETCD_INITIAL_ADVERTISE_PEER_URLS /etc/etcd/etcd.conf
```

3. ノードが以前のバージョンで etcd クラスターに含まれていた場合には、以前の etcd データを削除します。

```
# rm -Rf /var/lib/etcd/*
```

4. etcd が正しく実行されている etcd ホストで、新しいメンバーを追加します。

```
# etcdctl3 member add *<name>* \
  --peer-urls="*<advertise_peer_urls>*"
```

このコマンドは一部の変数を出力します。以下に例を示します。

```
ETCD_NAME="master2"
ETCD_INITIAL_CLUSTER="master-0.example.com=https://192.168.55.8:2380"
ETCD_INITIAL_CLUSTER_STATE="existing"
```

5. 新しいホストの **/etc/etcd/etcd.conf** ファイルに、以前のコマンドからの値を追加します。

```
# vi /etc/etcd/etcd.conf
```

6. クラスターに参加するノードで etcd サービスを起動します。

```
# systemctl start etcd.service
```

7. エラーメッセージの有無を確認します。

```
# master-logs etcd etcd
```

8. 全ノードを追加したら、クラスターのステータスと正常性を確認します。

```
# etcdctl3 endpoint health --
endpoints="https://<etcd_host1>:2379,https://<etcd_host2>:2379,https://<etcd_host3>:2379"
```

```
https://master-0.example.com:2379 is healthy: successfully committed proposal: took =
1.423459ms
https://master-1.example.com:2379 is healthy: successfully committed proposal: took =
1.767481ms
https://master-2.example.com:2379 is healthy: successfully committed proposal: took =
1.599694ms

# etcdctl3 endpoint status --
endpoints="https://<etcd_host1>:2379,https://<etcd_host2>:2379,https://<etcd_host3>:2379"
https://master-0.example.com:2379, 40bef1f6c79b3163, 3.2.5, 28 MB, true, 9, 2878
https://master-1.example.com:2379, 1ea57201a3ff620a, 3.2.5, 28 MB, false, 9, 2878
https://master-2.example.com:2379, 59229711e4bc65c8, 3.2.5, 28 MB, false, 9, 2878
```

9. 残りのピアをクラスターに戻します。

38.2. 静的 POD の ETCD クォーラム (定足数) の復元

etcd に静的 Pod を使用するクラスターで etcd のクォーラム (定足数) を失う場合、以下の手順を実行します。

手順

1. etcd Pod を停止します。

```
mv /etc/origin/node/pods/etcd.yaml .
```

2. etcd ホスト上で新規クラスターを一時的に強制します。

```
$ cp /etc/etcd/etcd.conf etcd.conf.bak
$ echo "ETCD_FORCE_NEW_CLUSTER=true" >> /etc/etcd/etcd.conf
```

3. etcd Pod を再起動します。

```
$ mv etcd.yaml /etc/origin/node/pods/.
```

4. etcd Pod を停止し、**FORCE_NEW_CLUSTER** コマンドを削除します。

```
$ mv /etc/origin/node/pods/etcd.yaml .
$ rm /etc/etcd/etcd.conf
$ mv etcd.conf.bak /etc/etcd/etcd.conf
```

5. etcd Pod を再起動します。

```
$ mv etcd.yaml /etc/origin/node/pods/.
```

第39章 OPENSHIFT SDN のトラブルシューティング

39.1. 概要

SDN のドキュメント で説明されているように、あるコンテナーから別のコンテナーへトラフィックを適切に渡すために作成されるインターフェイスの複数のレイヤーがあります。接続の問題をデバッグするには、スタックの複数のレイヤーをテストして問題の原因を判別する必要があります。本書は複数のレイヤーを調べて問題を特定し、解決するのに役立ちます。

問題の原因の一部は OpenShift Container Platform が複数の方法で設定でき、ネットワークが複数の異なる場所で正しく設定されない可能性がある点にあります。本書では、いくつかのシナリオを使用しますが、これらのシナリオは大半のケースに対応していることが予想されます。実際に生じている問題がこれらのシナリオで扱われていない場合には、導入されている各種のツールおよび概念を使用してデバッグ作業を行うことができます。

39.2. 用語

クラスター

クラスター内の一連のマシン。つまり、マスターとノードになります。

マスター

OpenShift Container Platform クラスターのコントローラーです。マスターはクラスター内のノードではない場合があり、そのため Pod への IP 接続がない場合があることに注意してください。

ノード

Pod をホストできる OpenShift Container Platform を実行するクラスター内のホストです。

Pod

OpenShift Container Platform によって管理される、ノード上で実行されるコンテナーのグループです。

Service

1 つ以上の Pod でサポートされる、統一ネットワークインターフェイスを表す抽象化です。

ルーター

複数の URL とパスを OpenShift Container Platform サービスにマップし、外部トラフィックがクラスターに到達できるようにする web プロキシーです。

ノードアドレス

ノードの IP アドレスです。これはノードが割り当てられるネットワークの所有者によって割り当てられ、管理されます。クラスター内の任意のノード (マスターおよびクライアント) からアクセスできる必要があります。

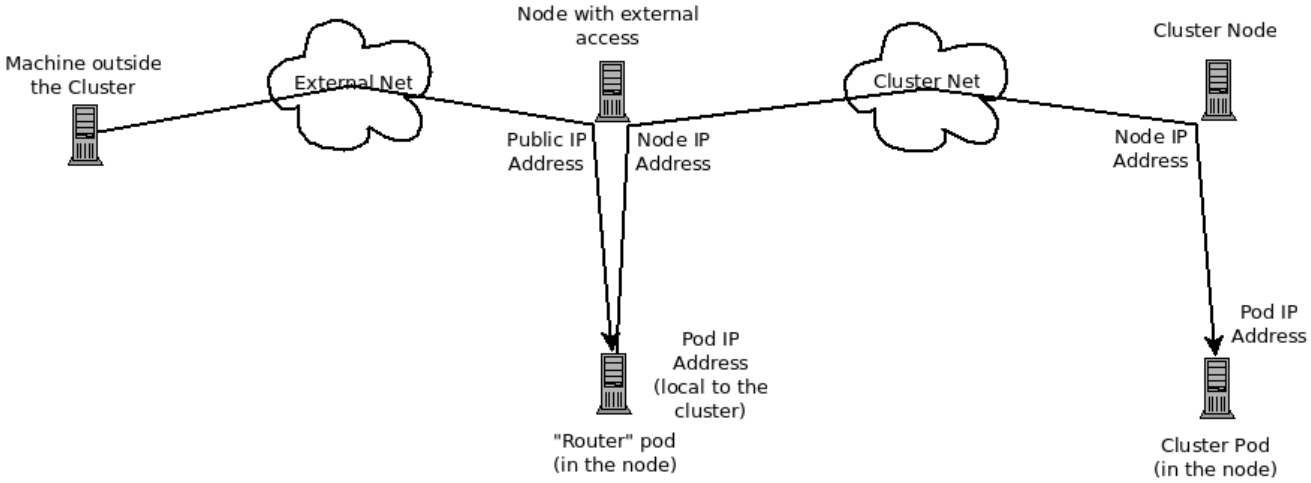
Pod アドレス

Pod の IP アドレスです。これらは OpenShift Container Platform によって割り当てられ、管理されます。デフォルトで、これらは 10.128.0.0/14 ネットワーク (または古いバージョンでは 10.1.0.0/16) から割り当てられます。クライアントノードからのみアクセスできます。

サービスアドレス

サービスを表す IP アドレスで、内部で Pod アドレスにマップされます。これらは OpenShift Container Platform によって割り当てられ、管理されます。デフォルトで、これらは 172.30.0.0/16 ネットワークから割り当てられます。クライアントノードからのみアクセスできます。

以下の図は、外部アクセスに関係するすべての設定部分を示しています。



39.3. HTTP サービスへの外部アクセスのデバッグ

クラスター外のマシンを使用している場合で、クラスターで提供されるリソースにアクセスしようとしている場合、パブリック IP アドレスでリッスンし、クラスター内のトラフィックをルーティングする Pod でプロセスが実行されている必要があります。この場合、OpenShift Container Platform ルーターは、HTTP、HTTPS (SNI を使用)、WebSocket または TLS (SNI を使用) について使用できます。

クラスター外より HTTP サービスにアクセスできないことを想定し、障害が発生しているマシンのコマンドラインを使って問題を再現します。以下を実行します。

```
curl -kv http://foo.example.com:8000/bar    # But replace the argument with your URL
```

成功する場合は、正しい場所からバグを再現しているかどうかを確認します。サービスに機能する Pod と機能しない Pod が含まれる可能性もあります。したがって、「ルーターのデバッグ」 セクションを参照してください。

失敗した場合は、IP アドレスに対して DNS 名を解決します (ないことを想定します)。

```
dig +short foo.example.com              # But replace the hostname with yours
```

IP アドレスが返されない場合は、DNS をトラブルシューティングする必要がありますが、これについては本書では扱いません。

重要

返される IP アドレスがルーターを実行するルーターであることを確認します。そうでない場合は、DNS を修正します。

次に **ping -c address** および **tracepath address** を使用して、ルーターホストに到達できることを確認します。それらが ICMP パケットに応答しない場合もあり、この場合はそれらのテストは失敗しますが、ルーターマシンにはアクセスできる場合があります。この場合、コマンドを使ってルーターのポートに直接アクセスしてみます。

```
telnet 1.2.3.4 8000
```

以下が表示される場合があります。

```
Trying 1.2.3.4...
Connected to 1.2.3.4.
Escape character is '^]'.
```

この場合、IP アドレスのポートでリッスンしているものがあることを示しています。これは適切と言えます。**ctrl-]** を押してから **enter** キーを押し、**close** と入力して telnet を終了します。「ルーターのデバッグ」 セクションに移動して、ルーター上の他の内容を確認します。

または、以下が表示される可能性があります。

```
Trying 1.2.3.4...
telnet: connect to address 1.2.3.4: Connection refused
```

これは、ルーターがそのポートでリッスンしていないことを示します。ルーターの設定方法における追加のポイントについては、「ルーターのデバッグ」 セクションを参照してください。

または、以下が表示される場合があります。

```
Trying 1.2.3.4...
  telnet: connect to address 1.2.3.4: Connection timed out
```

これは、IP アドレス上のいずれとも通信できないことを示します。ルーティング、ファイアウォールを確認し、IP アドレスでリッスンしているルーターがあることを確認します。ルーターをデバッグするには、「ルーターのデバッグ」 セクションを参照してください。IP ルーティングおよびファイアウォールの問題については、本書では扱いません。

39.4. ルーターのデバッグ

IP アドレスを使用し、そのマシンに対して **ssh** を実行してルーターソフトウェアがそのマシン上で実行されており、正しく設定されていることを確認する必要があります。ここで **ssh** を実行し、管理者の OpenShift Container Platform 認証情報を取得します。

注記

管理者の認証情報がある場合でも デフォルトシステムユーザー の **system:admin** としてログインしていない場合は、認証情報が CLI 設定ファイル にある限り、いつでもこのユーザーとしてログインし直すことができます。以下のコマンドはログインを実行し、デフォルト のプロジェクトに切り替えます。

```
$ oc login -u system:admin -n default
```

ルーターが実行されていることを確認します。

```
# oc get endpoints --namespace=default --selector=router
NAMESPACE   NAME            ENDPOINTS
default     router          10.128.0.4:80
```

このコマンドが失敗する場合、OpenShift Container Platform 設定は破損しています。この設定の修正については、本書では扱われません。

1 つ以上のルーターエンドポイントが一覧表示されますが、エンドポイント IP アドレスはクラスター内の Pod アドレスの 1 つであるため、それらが指定の外部 IP アドレスでマシン上で実行されているかどうかを識別することはできません。ルーターホスト IP アドレスの一覧を取得するには、以下を実行し

ます。

```
# oc get pods --all-namespaces --selector=router --template='{{range .items}}HostIP:
{{.status.hostIP}} PodIP: {{.status.podIP}}{{"\n"}}{{end}}'
HostIP: 192.168.122.202  PodIP: 10.128.0.4
```

外部アドレスに対応するホスト IP が表示されるはずです。表示されない場合は、ルーターのドキュメント を参照し、(アフィニティーを適切に設定して) 適切なノードで実行されるようにルーター Pod を設定するか、またはルーターが実行されている IP アドレスに一致するよう DNS を更新します。

(本書の) この時点では、ノードでルーター Pod を実行しても HTTP 要求を機能させることはできません。まず、ルーターが外部 URL を正しいサービスにマップしていること、またそれが機能している場合は、そのサービスの詳細を調べてすべてのエンドポイントがアクセス可能であることを確認する必要があります。

OpenShift Container Platform が認識するすべてのルートを一覧表示します。

```
# oc get route --all-namespaces
NAME              HOST/PORT        PATH     SERVICE       LABELS    TLS TERMINATION
route-unsecured   www.example.com  /test    service-name
```

URL のホスト名およびパスが返されるルートの一覧のいずれにも一致しない場合はルートを追加する必要があります。ルーターのドキュメント を参照してください。

ルートが存在する場合、エンドポイントへのアクセスをデバッグする必要があります。これはサービスに関する問題をデバッグしている場合と同様のプロセスです。 そのため、次の「サービスのデバッグ」 セクションに進んでください。

39.5. サービスのデバッグ

クラスター内からサービスと通信できない場合 (サービスが直接通信できないか、またはルーターを使用していてクラスターに入るまですべてが正常に機能している場合)、サービスに関連付けられているエンドポイントを判別し、それらをデバッグする必要があります。

最初にサービスを取得します。

```
# oc get services --all-namespaces
NAMESPACE   NAME            LABELS                                SELECTOR              IP(S)
PORT(S)
default     docker-registry  docker-registry=default               docker-registry=default
172.30.243.225  5000/TCP
default     kubernetes      component=apiserver,provider=kubernetes  <none>               172.30.0.1
443/TCP
default     router          router=router                         router=router         172.30.213.8    80/TCP
```

サービスが一覧に表示されます。表示されない場合は、サービス を定義する必要があります。

サービス出力に一覧表示される IP アドレスは Kubernetes サービス IP アドレスであり、これは Kubernetes がサービスをサポートする Pod のいずれかにマップするものです。このの IP アドレスと通信できるはずですが、通信できたとしても、すべての Pod にアクセスできる訳ではありません。また、通信できない場合もすべての Pod がアクセスできない訳ではありません。これは kubeproxy が接続している 1 つ の IP アドレスのステータスのみを示しています。

サービスをテストします。ノードのいずれかより以下を実行します。

```
curl -kv http://172.30.243.225:5000/bar                # Replace the argument with your service IP
address and port
```

次にサービスをサポートしている Pod を見つけます (**docker-registry** を破損したサービスの名前に置き換えます)。

```
# oc get endpoints --selector=docker-registry
NAME              ENDPOINTS
docker-registry   10.128.2.2:5000
```

ここではエンドポイントは 1 つだけであることを確認できます。そのため、サービステストが成功し、ルーターテストに成功した場合には、極めて稀なことが生じている可能性があります。ただし、複数のエンドポイントがあるか、またはサービステストが失敗した場合には、エンドポイントごとに以下を試行します。機能していないエンドポイントを特定できたら、次のセクションに進みます。

最初に、それぞれのエンドポイントをテストします (適切なエンドポイント IP、ポートおよびパスを持つように URL を変更します)。

```
curl -kv http://10.128.2.2:5000/bar
```

これが機能する場合は、次のエンドポイントをテストします。失敗した場合はその情報をメモしておきます。次のセクションでその原因を判別します。

すべてが失敗した場合は、ローカルノードが機能していない可能性があります。その場合は、「ローカルネットワークのデバッグ」 セクションに移動してください。

すべてが機能する場合は、「Kubernetes のデバッグ」 セクションに移動して、サービス IP アドレスが機能しない理由を判別します。

39.6. ノード間通信のデバッグ

機能していないエンドポイントの一覧を使用して、ノードに対する接続をテストする必要があります。

1. すべてのノードに予想される IP アドレスがあることを確認します。

```
# oc get hostsubnet
NAME                  HOST                  HOST IP         SUBNET
rh71-os1.example.com  rh71-os1.example.com  192.168.122.46  10.1.1.0/24
rh71-os2.example.com  rh71-os2.example.com  192.168.122.18  10.1.2.0/24
rh71-os3.example.com  rh71-os3.example.com  192.168.122.202  10.1.0.0/24
```

   DHCP を使用している場合はそれらが変更されている可能性があります。ホスト名、IP アドレス、およびサブネットが予想される内容に一致していることを確認します。ノードの詳細が変更されている場合は、**oc edit hostsubnet** を使用してエントリーを訂正します。

2. ノードアドレスおよびホスト名が正しいことを確認した後に、エンドポイント IP およびノード IP を一覧表示します。

```
# oc get pods --selector=docker-registry \
    --template='{{range .items}}HostIP: {{.status.hostIP}}   PodIP: {{.status.podIP}}{{end}}
{{"\n"}}'

HostIP: 192.168.122.202   PodIP: 10.128.0.4
```

3. 以前にメモしたエンドポイント IP アドレスを見つけ、これを **PodIP** エントリーで検索し、対応する **HostIP** アドレスを見つけます。次に、**HostIP** からのアドレスを使用してノードホストレベルで接続をテストします。

    - **ping -c 3 <IP_address>**: 応答がないことは、中間ルーターが ICMP トラフィックを消費している可能性があることを意味しています。

    - **tracepath <IP_address>**: ICMP パケットがすべてのホップによって返される場合、ターゲットにつながる IP ルートを表示します。
    tracepath と **ping** の両方が失敗する場合、ローカルまたは仮想ネットワークの接続の問題を探します。

4. ローカルネットワークの場合は、以下を確認します。

    - 追加設定なしの状態のパケットのターゲットアドレスへのルートを確認します。

        ```
        # ip route get 192.168.122.202
          192.168.122.202 dev ens3  src 192.168.122.46
            cache
        ```

        上記の例では、ソースアドレスが **192.168.122.46** の **ens3** という名前のインターフェイスから、ターゲットに直接つながります。これが予想される結果である場合は **ip a show dev ens3** を使用してインターフェイスの詳細を取得し、このインターフェイスが予想されるインターフェイスであることを確認します。

        または、結果が以下になる可能性もあります。

        ```
        # ip route get 192.168.122.202
          1.2.3.4 via 192.168.122.1 dev ens3  src 192.168.122.46
        ```

        これは、正しくルーティングされるために **via** 値をパススルーします。トラフィックが正しくルーティングされていることを確認します。ルートトラフィックのデバッグについては、本書では扱われません。

ノード間ネットワークの他のデバッグオプションについては、以下を確認して解決できます。

- 両端にイーサネットリンクはありますか ?**ethtool <network_interface>** の出力で、**Link detected: yes** を検索します。

- デュプレックス設定とイーサネット速度は、両端で正しく設定されていますか ?残りの **ethtool <network_interface>** 情報を確認します。

- ケーブルは適切にプラグインされていますか ?正しいポートにプラグインされていますか ?

- スイッチは適切に設定されていますか ?

ノード間設定が適切であることを確認した後は、両サイドで SDN 設定を確認する必要があります。

39.7. ローカルネットワークのデバッグ

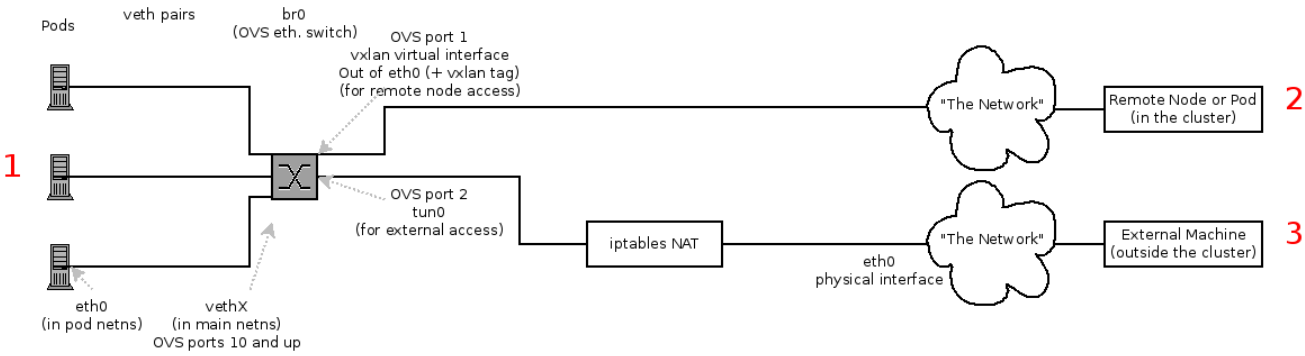
ここで通信できないものの、ノード間通信が設定された 1 つ以上のエンドポイントの一覧が表示されます。それぞれのエンドポイントについて問題点を特定する必要がありますが、まずは SDN が複数の異なる Pod についてノードでネットワークをどのように設定しているかについて理解する必要があります。

39.7.1. ノードのインターフェイス

以下は OpenShift SDN が作成するインターフェイスです。

- **br0**: コンテナーが割り当てられる OVS ブリッジデバイスです。OpenShift SDN は、このブリッジにサブネット以外のフロールールも設定します。
- **tun0**: OVS 内部ポート (**br0** のポート 2) です。これにはクラスターサブネットゲートウェイアドレスが割り当てられ、外部ネットワークアクセスに使用されます。OpenShift SDN はクラスターサブネットから NAT 経由で外部ネットワークにアクセスできるように **netfilter** およびルートルールを設定します。
- **vxlan_sys_4789**: OVS VXLAN デバイス (**br0** のポート 1) です。これはリモートノードのコンテナーへのアクセスを提供します。OVS ルールでは **vxlan0** として参照されます。
- **vethX** (メイン netns 内): Docker netns における **eth0** の Linux 仮想イーサネットのピアです。これは他のポートのいずれかの OVS ブリッジに割り当てられます。

39.7.2. ノード内の SDN フロー



アクセスしようとしているもの (またはアクセスされるもの) によってパスは異なります。SDN が (ノード内に) で接続する場所は 4 カ所あります。それらには上記の図で赤のラベルが付けられています。

- **Pod:** トラフィックは同じマシンのある Pod から別の Pod に移動します (1 から他の 1 へ)。
- リモートノード **(または Pod):** トラフィックは同じクラスター内のローカル Pod からリモートノードまたは Pod に移動します (1 から 2 へ)。
- 外部マシン: トラフィックはローカル Pod からクラスター外に移動します (1 から 3 へ)。

当然のこととして、トラフィックはこれらと反対方向でも移動します。

39.7.3. デバッグ手順

39.7.3.1. IP 転送は有効にされているか？

sysctl net.ipv4.ip_forward が 1 に設定されていること (およびホストが仮想マシンであるかどうか) を確認します。

39.7.3.2. ルートは正しく設定されているか？

ip route でルートテーブルを確認します。

```
# ip route
```

```
default via 192.168.122.1 dev ens3
10.128.0.0/14 dev tun0  proto kernel  scope link                    # This sends all pod traffic into OVS
10.128.2.0/23 dev tun0  proto kernel  scope link  src 10.128.2.1      # This is traffic going to local
pods, overriding the above
169.254.0.0/16 dev ens3  scope link  metric 1002                     # This is for Zeroconf (may not be
present)
172.17.0.0/16 dev docker0  proto kernel  scope link  src 172.17.42.1    # Docker's private IPs... used
only by things directly configured by docker; not OpenShift
192.168.122.0/24 dev ens3  proto kernel  scope link  src 192.168.122.46 # The physical interface on
the local subnet
```

10.128.x.x 行が表示されるはずです (Pod ネットワークが設定内でデフォルト範囲に設定されていることを前提とします)。これが表示されない場合は、OpenShift Container Platform ログを確認します (「ログの読み取り」 セクションを参照してください)。

39.7.4. Open vSwitch (OVS) は正しく設定されているか？

OVS Pod のいずれかで **ovs-vsctl** および **ovs-ofctl** コマンドを実行する必要があります。

OVS Pod を一覧表示するには、以下のコマンドを入力します。

```
$ oc get pod -n openshift-sdn -l app=ovs
```

両サイドで Open vSwitch ブリッジを確認します。**<ovs_pod_name>** を OVS Pod のいずれかの名前に置き換えます。

```
$ oc exec -n openshift-sdn <ovs_pod_name> -- ovs-vsctl list-br
br0
```

上記のコマンドは **br0** を返すはずです。

OVS が認識するすべてのポートを一覧表示できます。

```
$ oc exec -n openshift-sdn <ovs_pod_name> -- ovs-ofctl -O OpenFlow13 dump-ports-desc br0
OFPST_PORT_DESC reply (OF1.3) (xid=0x2):
 1(vxlan0): addr:9e:f1:7d:4d:19:4f
     config:     0
     state:      0
     speed: 0 Mbps now, 0 Mbps max
 2(tun0): addr:6a:ef:90:24:a3:11
     config:     0
     state:      0
     speed: 0 Mbps now, 0 Mbps max
 8(vethe19c6ea): addr:1e:79:f3:a0:e8:8c
     config:     0
     state:      0
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 LOCAL(br0): addr:0a:7f:b4:33:c2:43
     config:     PORT_DOWN
     state:      LINK_DOWN
     speed: 0 Mbps now, 0 Mbps max
```

とくにアクティブなすべての Pod の **vethX** デバイスがポートとして表示されるはずです。

次に、そのブリッジに設定されているフローを一覧表示します。

```
$ oc exec -n openshift-sdn <ovs_pod_name> --  ovs-ofctl -O OpenFlow13 dump-flows br0
```

ovs-subnet または **ovs-multitenant** プラグインのどちらを使用しているかに応じて結果は若干異なりますが、以下のような一般的な設定を確認することができます。

1. すべてのリモートノードには **tun_src=<node_IP_address>** に一致するフロー (ノードからの着信 VXLAN トラフィック) およびアクション **set_field:<node_IP_address>->tun_dst** を含む別のフロー (ノードへの発信 VXLAN トラフィック) が設定されている必要があります。

2. すべてのローカル Pod には **arp_spa=<pod_IP_address>** および **arp_tpa=<pod_IP_address>** に一致するフロー (Pod の着信および発信 ARP トラフィック) と、**nw_src=<pod_IP_address>** および **nw_dst=<pod_IP_address>** に一致するフロー (Pod の着信および発信 IP トラフィック) が設定されている必要があります。

フローがない場合は、「ログの読み取り」 セクションを参照してください。

39.7.4.1. iptables 設定に誤りがないか？

iptables-save の出力をチェックし、トラフィックにフィルターを掛けていないことを確認します。OpenShift Container Platform は通常の操作時に iptables ルールを設定するため、ここにエントリーが表示されていても不思議なことではありません。

39.7.4.2. 外部ネットワークは正しく設定されているか？

外部ファイアウォール (ある場合) を確認し、ターゲットアドレスへのトラフィックを許可するかどうかを確認します (これはサイトごとに異なるため、本書では扱われません)。

39.8. 仮想ネットワークのデバッグ

39.8.1. 仮想ネットワークのビルドに障害が発生している

仮想ネットワーク (例: OpeStack) を使用して OpenShift Container Platform をインストールしている場合で、ビルドに障害が発生している場合、ターゲットノードホストの最大伝送単位 (MTU: maximum transmission unit) はプライマリーネットワークインターフェイス (例: **eth0**) の MTU との互換性がない可能性があります。

ビルドが正常に完了するには、データをノードホスト間で渡すために SDN の MTU が eth0 ネットワークの MTU よりも小さくなければなりません。

1. **ip addr** コマンドを実行してネットワークの MTU を確認します。

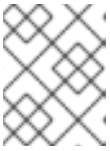
```
# ip addr
---
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
qlen 1000
    link/ether fa:16:3e:56:4c:11 brd ff:ff:ff:ff:ff:ff
    inet 172.16.0.0/24 brd 172.16.0.0 scope global dynamic eth0
       valid_lft 168sec preferred_lft 168sec
    inet6 fe80::f816:3eff:fe56:4c11/64 scope link
       valid_lft forever preferred_lft forever
---
```

上記のネットワークの MTU は 1500 です。

2. ノード設定の MTU はネットワーク値よりも小さくなければなりません。ターゲットに設定されたノードホストの **mtu** を確認します。

```
# $ oc describe configmaps node-config-infra
...
networkConfig:
  mtu: 1450
  networkPluginName: company/openshift-ovs-subnet
...
```

上記のノード設定ファイルでは、**mtu** 値はネットワーク MTU よりも低くなるため、設定は不要になります。**mtu** 値がこれより高くなる場合はファイルを編集して、値をプライマリーネットワークインターフェイスの MTU よりも少なくとも 50 単位分下げてノードサービスを再起動します。これにより、より大きなパケットのデータをノード間で渡すことが可能になります。



注記

クラスターのノードを変更するには、ノード設定マップ を必要に応じて更新します。**node-config.yaml** ファイルは手動で変更しないようにしてください。

39.9. POD の EGRESS のデバッグ

Pod から外部サービスへのアクセスを試行する場合、以下の例のようになります。

```
curl -kv github.com
```

DNS が適切に解決されていることを確認します。

```
dig +search +noall +answer github.com
```

これにより、github サーバーの IP アドレスが返されるはずですが、正しいアドレスが返されていることを確認します。アドレスが返されない場合やお使いのマシンのいずれかのアドレスが返される場合、ローカル DNS サーバーのワイルドカードエントリーに一致している可能性があります。

これを修正するには、ワイルドカードエントリーを持つ DNS サーバーが **/etc/resolv.conf** の **nameserver** として一覧表示されていないことを確認するか、または ワイルドカードドメインが **search** 一覧に一覧表示されていないことを確認する必要があります。

正しい IP アドレスが返される場合、「ローカルネットワークのデバッグ」 の前述のデバッグに関するアドバイスに従ってください。通常、トラフィックはポート 2 の Open vSwitch から **iptables** ルールおよびルートテーブルを通過するはずです。

39.10. ログの読み取り

次を実行します: **journalctl -u atomic-openshift-node.service --boot | less**

Output of setup script: 行を検索します。'+' で始まるすべての行については、その下にスクリプト手順が記述されます。この部分で明らかなエラーがあるかどうかを調べます。

スクリプトを追ってみると、**Output of adding table=0** という行を見つけることができるはずです。これは OVS ルールであり、エラーは存在しないはずです。

39.11. KUBERNETES のデバッグ

iptables -t nat -L を確認して、サービスがローカルマシンで **kubeproxy** の適切なポートに NAT されていることを確認します。

警告

上記についてはまもなく全面的に変更されます... Kubeproxy は除去され、**iptables** のみのソリューションに置き換わります。

39.12. 診断ツールを使用したネットワークの問題の検出

クラスター管理者として診断ツールを実行し、共通するネットワークの問題を診断します。

```
# oc adm diagnostics NetworkCheck
```

診断ツールは、指定したコンポーネントのエラー状態をチェックする一連のチェックを実行します。詳細は、診断ツール のセクションを参照してください。

注記

現時点で、診断ツールでは IP フェイルオーバーの問題を診断できません。回避策として、スクリプトをマスターの https://raw.githubusercontent.com/openshift/openshift-sdn/master/hack/ipf-debug.sh で (またはマスターへのアクセスのある別のマシンから) 実行して役に立つデバッグ情報を生成できます。ただし、このスクリプトはサポート対象外です。

デフォルトで、 **oc adm diagnostics NetworkCheck** はエラーのログを **/tmp/openshift/** に記録します。これは **--network-logdir** オプションで設定できます。

```
# oc adm diagnostics NetworkCheck --network-logdir=<path/to/directory>
```

39.13. その他の注意点

39.13.1. ingress についての追加情報

- Kube: サービスを NodePort として宣言し、クラスター内のすべてのマシンでそのポートを要求し、kube-proxy およびサポートする Pod にルーティングします。https://kubernetes.io/docs/concepts/services-networking/service/#type-nodeport を参照してください (一部のノードは外部からアクセスできる必要があります)。
- Kube: LoadBalancer として宣言し、独自に 判別したオブジェクトが残りを実行します。
- OS/AE: いずれもルーターを使用します。

39.13.2. TLS ハンドシェイクのタイムアウト

Pod がデプロイに失敗する場合、docker ログで TLS ハンドシェイクのタイムアウトを確認します。

```
$ docker log <container_id>
...
[...] couldn't get deployment [...] TLS handshake timeout
...
```

この状態は通常はセキュアな接続を確立する際のエラーであり、このエラーは tun0 とプライマリーインターフェイス (例: eth0) 間の MTU 値の大きな違い (例: tun0 MTU が 1500 に対し eth0 MTU が 9000 (ジャンボフレーム) である場合) によって引き起こされる可能性があります。

39.13.3. デバッグについての他の注意点

- (Linux 仮想イーサネットペア) のピアインターフェイスは **ethtool -S ifname** で判別できます。
- ドライバータイプ: **ethtool -i ifname**

第40章 診断ツール

40.1. 概要

oc adm diagnostics コマンドは一連のチェックを実行し、ホストまたはクラスターのエラーの状態についてチェックします。とくに以下を実行します。

- デフォルトのレジストリーおよびルーターが実行中であり、正しく設定されていることを確認します。
- **ClusterRoleBindings** および **ClusterRoles** で、ベースポリシーとの整合性を確認します。
- すべてのクライアント設定コンテキストが有効で接続可能であることを確認します。
- SkyDNS が適切に機能しており、Pod に SDN 接続があることを確認します。
- ホストのマスターおよびノード設定を検証します。
- ノードが実行中で、利用可能であることを確認します。
- 既知のエラーについてホストログを分析します。
- systemd ユニットがホストに対して予想通りに設定されていることを確認します。

40.2. 診断ツールの使用

OpenShift Container Platform は複数の方法でデプロイできます。これらには以下が含まれます。

- ソースからのビルド
- VM イメージ内への組み込み
- コンテナーイメージとしての使用
- エンタープライズ RPM の使用

それぞれの方法は設定および環境に応じて適宜使い分けられます。環境についての想定内容を最小限にするために、診断ツールが **openshift** バイナリーに組み込まれており、OpenShift Container Platform サーバーまたはクライアント内で診断が実行されます。

(可能であればマスターホストで、クラスター管理者として) 診断ツールを使用するには、以下を実行します。

```
# oc adm diagnostics
```

これは利用可能なすべての診断を実行し、環境に適用されない手順を省略します。

名前別に特定の診断を実行するか、または問題に対応する際に使用する特定の診断を実行することができます。以下に例を示します。

```
$ oc adm diagnostics
```

診断ツールの各種オプションでは、有効な設定ファイルが必要になります。たとえば、**NodeConfigCheck** はノード設定が利用可能でない場合は実行されません。

診断ツールは、デフォルトで標準設定ファイルの場所を使用します。

- クライアント:
  - **$KUBECONFIG** 環境変数で示される。
  - ~/.kube/config file
- マスター:
  - /etc/origin/master/master-config.yaml
- ノード:
  - /etc/origin/node/node-config.yaml

--config, **--master-config**、および **--node-config** オプションで標準以外の場所を指定できます。設定ファイルが指定されない場合、関連する診断は省略されます。

利用可能な診断には以下が含まれます。

診断名	目的
AggregatedLogging	集約されたログ統合を使用して適切な設定および操作を確認します。
AnalyzeLogs	systemd サービスログで問題の有無を確認します。チェックの実行に設定ファイルは不要です。
ClusterRegistry	クラスターにビルドおよびイメージストリームの作業用のコンテナーイメージレジストリーがあることを確認します。
ClusterRoleBindings	デフォルトのクラスターロールバインディングが存在し、ベースポリシーに応じて予想されるサブジェクトが含まれることを確認します。
ClusterRoles	クラスターロールが存在し、ベースポリシーに応じて予想されるパーミッションが含まれることを確認します。
ClusterRouter	クラスター内に有効なデフォルトルーターがあることを確認します。
ConfigContexts	クライアント設定の各コンテキストが完成したものであり、その API サーバーへの接続があることを確認します。
DiagnosticPod	アプリケーションの観点で診断を実行する Pod を作成します。 これは Pod 内の DNS が予想通りに機能しており、デフォルトサービスアカウントの認証情報がマスター API に対して正しく認証されることを確認します。

診断名	目的
EtcdWriteVolume	一定期間における etcd に対する書き込みのボリュームを確認し、操作およびキー別にそれらを分類します。この診断は他の診断と同じ速度で実行されず、かつ etcd への負荷を増えることから、とくに要求される場合にのみ実行されます。
MasterConfigCheck	このホストのマスター設定ファイルで問題の有無を確認します。
MasterNode	このホストで実行されているマスターがノードも実行していることを確認し、それがクラスター SDN のメンバーであることを確認します。
MetricsApiProxy	統合 Heapster メトリクスがクラスター API プロキシー経由でアクセス可能であることを確認します。
NetworkCheck	複数のノードで診断 Pod を作成し、一般的なネットワークの問題をアプリケーションまたは Pod の観点で診断します。マスターが Pod をノードにスケジュールできるものの、Pod に接続の問題がある場合にこの診断を実行します。このチェックは、Pod がサービス、他の Pod および外部ネットワークに接続できることを確認します。 エラーがある場合は、この診断は詳細な分析用としてローカルディレクトリー (デフォルトで **/tmp/openshift/**) に結果および取得されたファイルを保存します。ディレクトリーは **--network-logdir** フラグで指定することができます。
NodeConfigCheck	このホストのノード設定ファイルで問題の有無を確認します。
NodeDefinitions	マスター API で定義されたノードが利用可能な状態にあり、Pod をスケジュールできることを確認します。
RouteCertificateValidation	すべてのルート証明書で、拡張される検証で拒否される可能性のあるものがあるかどうかを確認します。
ServiceExternalIPs	マスター設定に基づいて無効にされている外部 IP を指定する既存サービスの有無を確認します。
UnitStatus	OpenShift Container Platform に関連して、このホストでユニットについての systemd ステータスを確認します。チェックの実行に設定ファイルは不要です。

40.3. サーバー環境における診断の実行

Ansible でデプロイされるクラスターは、OpenShift Container Platform クラスター内のノードに診断に関する追加の利点を提供します。これらには以下が含まれます。

- マスターおよびノード設定は標準的な場所にある設定ファイルに基づく。
- systemd ユニットがサーバーを管理するように設定される。
- マスターおよびノード設定ファイルはいずれも標準的な場所に置かれる。
- Systemd ユニットがクラスターでノードを管理するために作成され、設定される。
- すべてのコンポーネントが journald に対してログを記録する。

Ansible でデプロイされたクラスターで配置された設定ファイルのデフォルトの場所を維持することにより、フラグを使用せずに **oc adm diagnostics** を実行できます。設定ファイルにデフォルトの場所を使用していない場合は、**--master-config** および **--node-config** オプションを使用する必要があります。

```
# oc adm diagnostics --master-config=<file_path> --node-config=<file_path>
```

systemd ユニットおよび journald のログエントリーは現在のログ診断ロジックに必要です。他のデプロイメントタイプの場合、ログは単一ファイルに保存されるか、ノードとマスターを組み合わせるファイルに保存されるか、または標準出力 (stdout) に出力されます。ログエントリーで journald を使用しない場合、ログ診断は適切に機能せず、実行されません。

40.4. クライアント環境での診断の実行

診断ツールは通常のユーザーまたは **cluster-admin** として実行でき、これは実行の際に使用するアカウントに付与されたレベルのパーミッションを使って実行します。

通常のアクセスを持つクライアントはマスターへの接続を診断し、診断 Pod を実行することができます。複数のユーザーまたはマスターが設定されている場合、接続のテストはそれらすべてを対象に実行されますが、診断 Pod は現行ユーザー、サーバー、またはプロジェクトに対してのみ実行されます。

cluster-admin アクセスを持つクライアントは、ノード、レジストリー、およびルーターなどのインフラストラクチャーのステータスを診断できます。いずれの場合も、**oc adm diagnostics** を実行すると、標準の場所で標準のクライアント設定ファイルを検索し、利用可能な場合はこれを使用できます。

40.5. ANSIBLE ベースのヘルスチェック

追加のヘルスチェックは、OpenShift Container Platform クラスターのインストールおよび管理に使用する Ansible ベースのツール で利用できます。この正常性チェックでは、現行の OpenShift Container Platform インストールによくあるデプロイメントの問題を報告できます。

これらのチェックは、**ansible-playbook** コマンドの使用 (クラスターインストール で使用されるのと同じ方式) によるか、または **openshift-ansible** の コンテナー化されたバージョン として実行できます。**ansible-playbook** 方式については、チェックは **openshift-ansible** RPM パッケージを使って行われます。コンテナー化方式の場合は、**openshift3/ose-ansible** コンテナーイメージが Red Hat Container レジストリー 経由で配布されます。各方式の使用例については、後続のセクションで説明されます。

以下のヘルスチェックは、デプロイされた OpenShift Container Platform クラスターを対象に、指定された **health.yml** playbook を使用して Ansible インベントリーファイルに対して実行されることが意図されている診断タスクのセットのこと指します。

警告

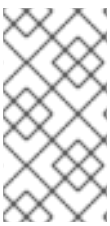
ヘルスチェック Playbook が環境に変更を加える可能性があるため、これらの Playbook は Ansible を使ってデプロイされたクラスターで、デプロイ時に使用したものと同じインベントリーファイルを使う場合にのみ使用できます。これらの変更には、チェックで必要な情報を収集できるように依存関係をインストールすることに関連するものです。そのような状況では、**docker** またはネットワーク設定などの追加のシステムコンポーネントは、現在の状態がインベントリーファイルの設定と異なる場合に変更される可能があります。これらのヘルスチェックは、使用するインベントリーファイルが既存のクラスター設定に変更を加えないことが予想される場合にのみ実行してください。

表40.1 正常性診断チェック

チェック名	目的
etcd_imagedata_size	このチェックは、etcd クラスターの OpenShift Container Platform イメージデータの合計サイズを測定します。このチェックは計算されたサイズがユーザー定義の制限を超える場合に失敗します。制限が指定されない場合、このチェックはイメージデータのサイズが etcd クラスターで現在使用されている領域の 50 % 以上になる場合に失敗します。 このチェックに失敗することは、etcd の多くの領域が OpenShift Container Platform イメージデータによって使用されていることを示し、これにより、最終的には etcd クラスターがクラッシュする可能性があります。 ユーザー定義の制限は **etcd_max_image_data_size_bytes** 変数を渡すことで設定できます。たとえば、**etcd_max_image_data_size_bytes=40000000000** を設定する場合、etcd に保存されるイメージデータの合計サイズが 40 GB を超えるとチェックが失敗します。

チェック名	目的
etcd_traffic	このチェックは、etcd ホストの通常よりも高いレベルのトラフィックを検知します。etcd 期間の警告と共に **journalctl** ログエントリーが見つかる場合に失敗します。 etcd パフォーマンスを強化する方法についての詳細は、Recommended Practices for OpenShift Container Platform etcd Hosts および Red Hat ナレッジベース を参照してください。
etcd_volume	このチェックにより、etcd クラスターのボリューム使用がユーザー指定の最大しきい値を超えないようにできます。最大しきい値が指定されていない場合、デフォルトは合計ボリュームサイズの **90%** に設定されます。 ユーザー定義の制限は、**etcd_device_usage_threshold_percent** 変数を渡すことで設定できます。
docker_storage	docker デーモン (ノードおよびコンテナー化されたインストール) に依存するホストでのみ実行されます。docker の合計使用量がユーザー定義制限を超えないこと確認します。ユーザー定義の制限が設定されていない場合、docker 使用量の最大しきい値のデフォルトは利用可能な合計サイズの 90% になります。 合計パーセントの使用量についてのしきい値の制限は、**max_thinpool_data_usage_percent=90** などのようにインベントリーファイルの変数で設定できます。 また、このチェックは docker のストレージが サポートされる設定 を使用していることを確認します。
curator、elasticsearch、fluentd、kibana	この一連のチェックは、Curator、Kibana、Elasticsearch、および Fluentd Pod がデプロイされており、これらが **running** 状態であることを検証し、接続を制御ホストと公開される Kibana URL 間で確立できることを検証します。これらのチェックは、**openshift_logging_install_logging** インベントリー変数が **true** に設定されている場合にのみ実行され、それらが クラスターロギング が有効にされているデプロイメントで実行されるようにします。

チェック名	目的
logging_index_time	このチェックは、ロギングスタックデプロイメントにおけるログ作成から Elasticsearch によるログ集計までの通常の時間差よりも値が高くなるケースを検知します。新規のログエントリーがタイムアウト内 (デフォルトでは 30 秒内) に Elasticserach によってクエリーされない場合に失敗します。このチェックはロギングが有効にされている場合にのみ実行されます。 ユーザー定義のタイムアウトは、**openshift_check_logging_index_timeout_seconds** 変数を渡すことで設定できます。たとえば、**openshift_check_logging_index_timeout_seconds=45** を設定すると、新規作成されるログエントリーが 45 秒を経過しても Elasticsearch でクエリーされない場合に失敗します。
sdn	このチェックは、OpenShift Container Platform SDN の以下のクラスターレベルの診断を実行します。 ● マスターホストが kubelets に接続できることを検証します。 ● ノードがパケットを相互にルート指定できることを検証します。 ● ノードアドレスを検証します。 ● HostSubnet オブジェクトを検証します。 **openshift_checks_output_dir** 変数を **ansible-playbook** コマンドで指定する場合、このチェックにより、OpenShift Container Platform API からのネットワーク関連のオブジェクトが、指定されるディレクトリーにあるログ、OVS フロー、iptables ルールその他のネットワーク設定情報と共に保存されます。変数の設定方法については、以下の **ansible-playbook** コマンドの使用例を参照してください。 このチェックは、**oc adm diagnostics** コマンドが診断 Pod をスケジュールできないか、または診断 Pod が問題のトラブルシューティングに必要な情報を十分に提供しない場合の Pod またはインフラストラクチャーの問題を診断するのに役立ちます。



注記

インストールプロセスの一部として実行されることが意図されている同様のチェックセットについては、Configuring Cluster Pre-install Checks を参照してください。証明書の有効期限をチェックするための別のチェックのセットについては、Redeploying Certificates を参照してください。

40.5.1. ansible-playbook によるヘルスチェックの実行

ansible-playbook コマンドを使用して **openshift-ansible** のヘルスチェックを実行するには、Playbook ディレクトリーに切り替え、クラスターのインベントリーファイルを指定し、**health.yml** Playbook を実行します。

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook -i <inventory_file> \
    playbooks/openshift-checks/health.yml
```

コマンドラインに変数を設定するには、**key=value** 形式の任意の必要な変数に **-e** フラグを組み込みます。以下に例を示します。

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook -i <inventory_file> \
    playbooks/openshift-checks/health.yml \
    -e openshift_check_logging_index_timeout_seconds=45 \
    -e etcd_max_image_data_size_bytes=40000000000
```

特定のチェックを無効にするには、Playbook を実行する前にインベントリーファイルのコンマ区切りのチェック名の一覧と共に変数 **openshift_disable_check** を組み込みます。以下に例を示します。

```
openshift_disable_check=etcd_traffic,etcd_volume
```

または、**ansible-playbook** コマンドの実行時に **-e openshift_disable_check=<check1>,<check2>** で変数として無効にするチェックを設定します。

40.5.2. Docker CLI でのヘルスチェックの実行

コンテナーで **openshift-ansible** Playbook を実行し、Docker CLI で **ose-ansible** イメージを実行できるホストでの Ansible のインストールおよび設定の手間を省くことができます。

以下を、コンテナーを実行する権限を持つ root 以外のユーザーとして実行します。

```
# docker run -u `id -u` \ 1
    -v $HOME/.ssh/id_rsa:/opt/app-root/src/.ssh/id_rsa:Z,ro \ 2
    -v /etc/ansible/hosts:/tmp/inventory:ro \ 3
    -e INVENTORY_FILE=/tmp/inventory \
    -e PLAYBOOK_FILE=playbooks/openshift-checks/health.yml \ 4
    -e OPTS="-v -e openshift_check_logging_index_timeout_seconds=45 -e
etcd_max_image_data_size_bytes=40000000000" \ 5
    openshift3/ose-ansible
```

1. これらのオプションにより、コンテナーは現行ユーザーと同じ UID で実行されます。 これは SSH キーをコンテナー内で読み取られるようにするようにパーミッションで必要になります (SSH プライベートキーはその所有者によってのみ読み取り可能であることが予想されます)。

2. SSH キーは、コンテナーを非 root ユーザーとして実行するなどの通常の使用では **/opt/app-root/src/.ssh** の下にマウントします。

3. **/etc/ansible/hosts** は、異なる場合はクラスターのインベントリーファイルの場所に切り替えます。このファイルは、コンテナーの **INVENTORY_FILE** 環境変数に基づいて使用される **/tmp/inventory** にバインドおよびマウントされます。

4 **PLAYBOOK_FILE** 環境変数は、コンテナー内の **/usr/share/ansible/openshift-ansible** に関連して **health.yml** playbook の場所に設定されます。

5 **-e key=value** 形式で単一の実行に必要な変数を設定します。

上記のコマンドでは、SSH キーは **:Z** オプションを使ってマウントされ、コンテナーが SSH キーを制限付き SELinux コンテキストから読み取れるようにします。このオプションを追加することは、元の SSH キーファイルのラベルが **system_u:object_r:container_file_t:s0:c113,c247** などに変更されることを意味しています。**:Z** についての詳細は、**docker-run(1)** man ページを参照してください。

重要

これらのボリュームマウント仕様に関連して予期しない影響が生じる可能性があります。たとえば、**$HOME/.ssh** ディレクトリーをマウント (したがってラベルを変更) する場合、**sshd** はパブリックキーにアクセスしてリモートログインを許可できなくなります。元のファイルラベルの変更を防ぐには、SSH キー (またはディレクトリー) のコピーをマウントします。

.ssh ディレクトリー全体をマウントすることは、以下に役立ちます。

- キーをホストに一致させたり、他の接続パラメーターを変更したりするために SSH 設定を使用することを許可します。
- ユーザーが **known_hosts** ファイルを指定し、SSH でホストキーを検証することを許可します。これはデフォルトの設定では無効にされていますが、**-e ANSIBLE_HOST_KEY_CHECKING=True** を **docker** コマンドラインに追加することにより、環境変数を使用してこれを再度有効にできます。

第41章 アプリケーションのアイドリング

41.1. 概要

OpenShift Container Platform 管理者は、アプリケーションをアイドリング状態にしてリソース消費を減らすことができます。これは、コストがリソース消費と関連付けられるパブリッククラウドにデプロイされている場合に役立ちます。

スケーラブルなリソースが使用されていない場合、OpenShift Container Platform はリソースを検出した後にそれらを 0 レプリカに設定してアイドリングします。ネットワークトラフィックがリソースに送信される場合、レプリカをスケールアップしてアイドリング解除を実行し、操作を続行します。

アプリケーションは複数のサービスやデプロイメント設定などの他のスケーラブルななリソースで設定されています。アプリケーションのアイドリングには、関連するすべてのリソースのアイドリングを実行することが関係します。

41.2. アプリケーションのアイドリング

アプリケーションのアイドリングには、サービスに関連付けられたスケーラブルなリソース (デプロイメント設定、レプリケーションコントローラーなど) を検索することが必要です。アプリケーションのアイドルリングには、サービスを検索してこれをアイドリング状態としてマークし、リソースを zero レプリカにスケールダウンすることが関係します。

oc idle コマンドを使用して 単一サービスのアイドリング を実行するか、または **--resource-names-file** オプションを使用して 複数サービスのアイドリング を実行できます。

41.2.1. 単一サービスのアイドリング

以下のコマンドを使用して単一サービスをアイドリングします。

```
$ oc idle <service>
```

41.2.2. 複数サービスのアイドリング

必要なサービスの一覧を作成し、**--resource-names-file** オプションを **oc idle** コマンドで使用することで複数サービスをアイドリングします。

これは、アプリケーションがプロジェクト内の一連のサービスにまたがる場合や、同じプロジェクト内で複数のアプリケーションを一括してアイドリングするため、複数サービスをスクリプトを併用してアイドリングする場合に役立ちます。

1. 複数サービスの一覧を含むテキストファイルを作成します (それぞれを各行に指定)。
2. **--resource-names-file** オプションを使用してサービスをアイドリングします。

```
$ oc idle --resource-names-file <filename>
```

注記

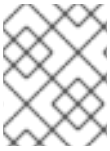
idle コマンドは単一プロジェクトに制限されます。クラスター全体でアプリケーションをアイドリングするには、各プロジェクトに対して idle コマンドをそれぞれ実行します。

41.3. アプリケーションのアイドリング解除

アプリケーションサービスは、ネットワークトラフィックを受信し、直前の状態に再びスケールアップすると再びアクティブになります。これには、サービスへのトラフィックとルートを通るトラフィックの両方が含まれます。

アプリケーションのアイドリング解除はリソースをスケールアップすることで手動で実行できます。たとえば、deploymentconfig をスケールアップするには、以下のコマンドを実行します。

```
$ oc scale --replicas=1 dc <deploymentconfig>
```



注記

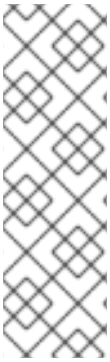
現時点で、ルーターによる自動アイドルリング解除はデフォルトの HAProxy ルーターのみでサポートされています。

第42章 クラスター容量の分析

42.1. 概要

クラスター管理者は、**hypercc cluster-capacity** ツールを使用して、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、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。

hypercc cluster-capacity ツール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。

hypercc cluster-capacity 分析ツールは、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、OpenShift Container Platform 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 内のジョブとして実行すると、介入なしに複数回実行することができます。

42.2. コマンドラインでのクラスター容量分析の実行

openshift-enterprise-cluster-capacity RPM パッケージをインストールして、ツールを取得します。コマンドラインでツールを実行するには、以下を実行します。

```
$ hypercc cluster-capacity --kubeconfig <path-to-kubeconfig> \
    --podspec <path-to-pod-spec>
```

--kubeconfig オプションは Kubernetes 設定ファイルを示し、**--podspec** オプションは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 仕様ファイルを示します。**podspec** はそのリソース要件を **limits** または **requests** として指定します。**hypercc cluster-capacity** 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。

Pod 仕様入力の例は以下の通りです。

```
apiVersion: v1
kind: Pod
metadata:
  name: small-pod
  labels:
    app: guestbook
    tier: frontend
spec:
  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
```

--verbose オプションを追加して、クラスター内の各ノードにスケジュールできる Pod 数についての詳細説明を出力できます。

```
$ hypercc cluster-capacity --kubeconfig <path-to-kubeconfig> \
    --podspec <path-to-pod-spec> --verbose
```

出力は以下のようになります。

```
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)
```

上記の例では、クラスターにスケジュールできる Pod の見積り数は 52 です。

42.3. POD 内のジョブとしてのクラスター容量分析の実行

クラスター容量ツールを Pod 内のジョブとして実行すると、ユーザーの介入なしに複数回実行できるという利点があります。クラスター容量ツールをジョブとして実行するには、**ConfigMap** を使用する必要があります。

1. クラスターロールを作成します。

```
$ cat << EOF| oc create -f -
kind: ClusterRole
apiVersion: v1
metadata:
  name: cluster-capacity-role
rules:
- apiGroups: [""]
  resources: ["pods", "nodes", "persistentvolumeclaims", "persistentvolumes", "services"]
  verbs: ["get", "watch", "list"]
EOF
```

2. サービスアカウントを作成します。

```
$ oc create sa cluster-capacity-sa
```

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:default:cluster-capacity-sa 1
```

1 サービスアカウントが **default** プロジェクトにない場合、**default** をプロジェクト名に置き換えます。

4. Pod 仕様を定義し、作成します。

```
apiVersion: v1
kind: Pod
metadata:
  name: small-pod
  labels:
    app: guestbook
    tier: frontend
spec:
  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
```

5. クラスター容量分析は、**cluster-capacity-configmap** という名前の **ConfigMap** 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。
ConfigMap を作成していない場合は、ジョブの作成前にこれを作成します。

```
$ oc create configmap cluster-capacity-configmap \
    --from-file=pod.yaml
```

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。

```
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: registry.redhat.io/openshift3/ose-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。

7. クラスター容量イメージを Pod のジョブとして実行します。

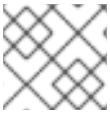
```
$ oc create -f cluster-capacity-job.yaml
```

8. 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 数を確認します。

```
$ oc logs jobs/cluster-capacity-job
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)
```

第43章 AWS でのクラスターの自動スケーラーの設定

Amazon Web Services (AWS) の OpenShift Container Platform クラスターで自動スケーラーを設定すると、アプリケーションのワークロードに対して弾力性を確保できます。自動スケーラーは Pod の実行にあたって十分な数のノードがアクティブであり、アクティブなノードの数が現在のデマンドに対応することを確認します。



注記

自動スケーラーは AWS でのみ実行できます。

43.1. OPENSHIFT CONTAINER PLATFORM 自動スケーラーについて

OpenShift Container Platform の自動スケーラーは、ノードの割り当てを保留にしている Pod 数を確認するために繰り返しチェックを実行します。Pod が割り当てを保留にしていて、自動スケーラーがその最大容量を達していない場合、現在の需要に対応するために新規ノードが継続的にプロビジョニングされます。需要が下がり、必要なノードが少なくなると、自動スケーラーは使用されていないノードを削除します。自動スケーラーをインストールした後に、その動作は自動化されます。そのため、必要な数のレプリカをデプロイメントに追加することのみが必要になります。

OpenShift Container Platform バージョン 3.11 では、自動スケーラーを Amazon Web Services (AWS) のみにデプロイできます。自動スケーラーは、自動スケーリンググループおよび起動設定などの一部の標準的な AWS オブジェクトを使用してクラスターのサイズを管理します。

自動スケーラーは以下のアセットを使用します。

自動スケーリンググループ

自動スケーリンググループはマシンセットの論理表現です。自動スケーリンググループは、実行する最小数のインスタンス、実行可能な最大数のインスタンスおよび、実行する必要のあるインスタンス数で設定します。必要な容量に対応するのに必要な数のインスタンスを起動すると、自動スケーリンググループが起動します。自動スケーリンググループは、インスタンスがゼロの状態で起動するように設定できます。

起動設定

起動設定は、インスタンスを起動するために自動スケーリングが使用するテンプレートです。起動設定を作成する際に、以下のような情報を指定します。

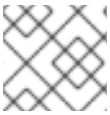
- ベースイメージを使用するための Amazon Machine Image (AMI) の ID
- **m4.large** などのインスタンスタイプ
- キーペア
- 1 つ以上のセキュリティーグループ
- 起動設定を適用するサブネット

OpenShift Container Platform Primed イメージ

自動スケーリンググループが新規インスタンスのプロビジョニングを実行する場合、それが起動するイメージに OpenShift Container Platform が準備されている必要があります。自動スケーリンググループはこのイメージを使用して、手動の介入なしにノードのブートストラップおよびクラスター内での登録を自動的に実行します。

43.2. PRIMED イメージの作成

自動スケーラーが使用する Primed イメージを自動的に作成するように Ansible Playbook を使用できます。既存の Amazon Web Services (AWS) クラスターの属性を指定する必要があります。



注記

すでに Primed イメージがある場合、新規のイメージを作成せずにこれを使用できます。

手順

OpenShift Container Platform クラスターを作成するために使用したホストで、Primed イメージを作成します。

1. ローカルホストに新規の Ansible インベントリーファイルを作成します。このファイルには、参加ノードで自動スケーリングを有効にするために **cloudprovider** フラグを割り当てる変数が必要です。これらの変数がないと、**build_ami.yml** Playbook は **openshift_cloud_provider** ロールを使用できません。

```
[OSEv3:children]
masters
nodes
etcd

[OSEv3:vars]
openshift_deployment_type=openshift-enterprise
ansible_ssh_user=ec2-user
openshift_clusterid=mycluster
ansible_become=yes
openshift_cloudprovider_kind=aws (1)
openshift_cloudprovider_aws_access_key=<aws_access_key> (2)
openshift_cloudprovider_aws_secret_key=<aws_secret_key> (3)

[masters]
[etcd]
[nodes]
```

(1) クラウドプロバイダーのタイプを指定します。

(2) クラウドプロバイダーの Identity and Access Management (IAM) アクセスキーを提供します。

(3) クラウドが提供する IAM シークレットキーを提供します。

2. ローカルホストにプロビジョニングファイル **build-ami-provisioning-vars.yaml** を作成します。

```
openshift_deployment_type: openshift-enterprise

openshift_aws_clusterid: mycluster (1)

openshift_aws_region: us-east-1 (2)

openshift_aws_create_vpc: false (3)

openshift_aws_vpc_name: production (4)
```

```
openshift_aws_subnet_az: us-east-1d 5

openshift_aws_create_security_groups: false 6

openshift_aws_ssh_key_name: production-ssh-key 7

openshift_aws_base_ami: ami-12345678 8

openshift_aws_create_s3: False 9

openshift_aws_build_ami_group: default 10

openshift_aws_vpc: 11
  name: "{{ openshift_aws_vpc_name }}"
  cidr: 172.18.0.0/16
  subnets:
    us-east-1:
    - cidr: 172.18.0.0/20
      az: "us-east-1d"

container_runtime_docker_storage_type: overlay2 12
container_runtime_docker_storage_setup_device: /dev/xvdb 13

# atomic-openshift-node service requires gquota to be set on the
# filesystem that hosts /var/lib/origin/openshift.local.volumes (OCP
# emptydir). Often is it not ideal or cost effective to deploy a vol
# for emptydir. This pushes emptydir up to the / filesystem. Base ami
# often does not ship with gquota enabled for /. Set this bool true to
# enable gquota on / filesystem when using Red Hat Cloud Access RHEL7
# AMI or Amazon Market RHEL7 AMI.
openshift_aws_ami_build_set_gquota_on_slashfs: true 14

rhsub_user: user@example.com 15
rhsub_pass: password 16
rhsub_pool: pool-id 17
```

1. 既存クラスターの名前を指定します。
2. 既存クラスターが現在実行されているリージョンを指定します。
3. **False** を指定して VPC の作成を無効にします。
4. クラスターが実行されている既存の VPC 名を指定します。
5. 既存のクラスターが実行されているサブネットの名前を指定します。
6. セキュリティーグループの作成を無効にするために **False** を指定します。
7. SSH アクセスに使用する AWS キー名を指定します。
8. Primed イメージのベースイメージとして使用する AMI イメージ ID を指定します。Red Hat® Cloud Access を参照してください。
9. S3 バケットの作成を無効にするために **False** を指定します。

10 セキュリティーグループ名を指定します。

11 既存クラスターが実行されている VPC サブネットを指定します。

12 **overlay2** を Docker ストレージタイプとして指定します。

13 LVM および **/var/lib/docker** ディレクトリーのマウントポイントを指定します。

14 Red Hat Cloud を使用する場合、このパラメーターの値を **true** に設定して、ファイルシステムで **gquota** を有効にします。

15 アクティブな OpenShift Container Platform サブスクリプションのある Red Hat アカウントのメールアドレスを指定します。

16 Red Hat アカウントのパスワードを指定します。

17 OpenShift Container Platform サブスクリプションのプール ID を指定します。クラスター作成時に使用したのと同じプール ID を使用できます。

3. **build_ami.yml** Paybook を実行して Primed イメージを生成します。

```
# ansible-playbook -i </path/to/inventory/file> \
    /usr/openshift-ansible/playbooks/aws/openshift-cluster/build_ami.yml \
    -e @build-ami-provisioning-vars.yaml
```

Playbook の実行後に、新規のイメージ ID または AMI が出力に表示されます。起動設定の作成時に生成した AMI を指定します。

43.3. 起動設定および自動スケーリンググループの作成

クラスターの自動スケーラーをデプロイする前に、Primed イメージを参照する Amazon Web Services (AWS) 起動設定および自動スケーリンググループを作成する必要があります。新規ノードが起動時に既存クラスターに自動的に参加するように起動設定を設定する必要があります。

前提条件

- AWS に OpenShift Container Platform クラスターをインストールします。
- Primed イメージを作成します。
- EFK スタックをクラスターにデプロイしている場合は、ノードラベルを **logging-infra-fluentd=true** に設定します。

手順

1. **bootstrap.kubeconfig** ファイルをマスターノードから生成し、これを作成します。

```
$ ssh master "sudo oc serviceaccounts create-kubeconfig -n openshift-infra node-
bootstrapper" > ~/bootstrap.kubeconfig
```

2. **user-data.txt** cloud-init ファイルを **bootstrap.kubeconfig** ファイルから作成します。

```
$ cat <<EOF > user-data.txt
#cloud-config
```

```
write_files:
- path: /root/openshift_bootstrap/openshift_settings.yaml
  owner: 'root:root'
  permissions: '0640'
  content: |
    openshift_node_config_name: node-config-compute
- path: /etc/origin/node/bootstrap.kubeconfig
  owner: 'root:root'
  permissions: '0640'
  encoding: b64
  content: |
    $(base64 ~/bootstrap.kubeconfig | sed '2,$s/^/    /')

runcmd:
- [ ansible-playbook, /root/openshift_bootstrap/bootstrap.yml]
- [ systemctl, restart, systemd-hostnamed]
- [ systemctl, restart, NetworkManager]
- [ systemctl, enable, atomic-openshift-node]
- [ systemctl, start, atomic-openshift-node]
EOF
```

3. 起動設定テンプレートを AWS S3 バケットにアップロードします。

4. AWS CLI を使用して起動設定を作成します。

```
$ aws autoscaling create-launch-configuration \
    --launch-configuration-name mycluster-LC \ 1
    --region us-east-1 \ 2
    --image-id ami-987654321 \ 3
    --instance-type m4.large \ 4
    --security-groups sg-12345678 \ 5
    --template-url https://s3-.amazonaws.com/.../yourtemplate.json \ 6
    --key-name production-key \ 7
```

1. 起動設定名を指定します。
2. イメージを起動するリージョンを指定します。
3. 作成した Primed イメージ AMI を指定します。
4. 起動するインスタンスのタイプを指定します。
5. 起動したイメージに割り当てるセキュリティーグループを指定します。
6. アップロードした起動設定テンプレートを指定します。
7. SSH キーペアの名前を指定します。

注記

テンプレートをエンコードする前に 16 KB 未満の場合は、AWS CLI を使用し、**--template-url** と **--user-data** を置き換えてテンプレートを指定できます。

5. AWS CLI を使用して自動スケーリンググループを作成します。

```
$ aws autoscaling create-auto-scaling-group \
    --auto-scaling-group-name mycluster-ASG \ 1
    --launch-configuration-name mycluster-LC \ 2
    --min-size 1 \ 3
    --max-size 6 \ 4
    --vpc-zone-identifier subnet-12345678 \ 5
    --tags ResourceId=mycluster-ASG,ResourceType=auto-scaling-
group,Key=Name,Value=mycluster-ASG-node,PropagateAtLaunch=true
ResourceId=mycluster-ASG,ResourceType=auto-scaling-
group,Key=kubernetes.io/cluster/mycluster,Value=true,PropagateAtLaunch=true
ResourceId=mycluster-ASG,ResourceType=auto-scaling-group,Key=k8s.io/cluster-
autoscaler/node-template/label/node-
role.kubernetes.io/compute,Value=true,PropagateAtLaunch=true 6
```

1. 自動スケーラーデプロイメントのデプロイ時に使用する自動スケーリンググループの名前を指定します。
2. 作成した起動設定の名前を指定します。
3. 自動スケーラーが維持するノードの最小数を指定します。1 つ以上のノードが必要です。
4. スケールグループが拡張できるノードの最大数を指定します。
5. クラスターが使用するのと同じサブネットの VPC サブネット ID を指定します。
6. 自動スケーリンググループタグが起動時のノードに伝播されるように文字列を指定します。

43.4. クラスターへの自動スケーラーコンポーネントのデプロイ

起動設定および自動スケーリンググループの作成後に、自動スケーラーコンポーネントをクラスターにデプロイできます。

前提条件

- AWS で OpenShift Container Platform クラスターをインストールします。
- Primed イメージを作成します。
- Primed イメージを参照する起動設定および自動スケーリンググループを作成します。

手順

自動スケーラーをデプロイするには、以下を実行します。

1. 自動スケーラーを実行するためにクラスターを更新します。
   a. クラスターを作成するために使用したインベントリーファイルに、以下のパラメーターを追加します。デフォルトは **/etc/ansible/hosts** です。

```
openshift_master_bootstrap_auto_approve=true
```

b. 自動スケーラーコンポーネントを取得するには、Playbook ディレクトリーに切り替えてから Playbook を再度実行します。

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook -i </path/to/inventory/file> \
    playbooks/openshift-master/enable_bootstrap.yml
```

c. **bootstrap-autoapprover** Pod が実行中であることを確認します。

```
$ oc get pods --all-namespaces | grep bootstrap-autoapprover
NAMESPACE             NAME                                READY     STATUS
RESTARTS   AGE
openshift-infra       bootstrap-autoapprover-0            1/1       Running   0
```

2. 自動スケーラーの namespace を作成します。

```
$ oc apply -f - <<EOF
apiVersion: v1
kind: Namespace
metadata:
  name: cluster-autoscaler
  annotations:
    openshift.io/node-selector: ""
EOF
```

3. 自動スケーラーのサービスアカウントを作成します。

```
$ oc apply -f - <<EOF
apiVersion: v1
kind: ServiceAccount
metadata:
  labels:
    k8s-addon: cluster-autoscaler.addons.k8s.io
    k8s-app: cluster-autoscaler
  name: cluster-autoscaler
  namespace: cluster-autoscaler
EOF
```

4. 必要なパーミッションをサービスアカウントに付与するためのクラスターロールを作成します。

```
$ oc apply -n cluster-autoscaler -f - <<EOF
apiVersion: v1
kind: ClusterRole
metadata:
  name: cluster-autoscaler
rules:
- apiGroups: 1
  - ""
  resources:
  - pods/eviction
  verbs:
  - create
  attributeRestrictions: null
```

```
- apiGroups:
  - ""
  resources:
  - persistentvolumeclaims
  - persistentvolumes
  - pods
  - replicationcontrollers
  - services
  verbs:
  - get
  - list
  - watch
  attributeRestrictions: null
- apiGroups:
  - ""
  resources:
  - events
  verbs:
  - get
  - list
  - watch
  - patch
  - create
  attributeRestrictions: null
- apiGroups:
  - ""
  resources:
  - nodes
  verbs:
  - get
  - list
  - watch
  - patch
  - update
  attributeRestrictions: null
- apiGroups:
  - extensions
  - apps
  resources:
  - daemonsets
  - replicasets
  - statefulsets
  verbs:
  - get
  - list
  - watch
  attributeRestrictions: null
- apiGroups:
  - policy
  resources:
  - poddisruptionbudgets
  verbs:
  - get
  - list
```

```
  - watch
  attributeRestrictions: null
EOF
```

1. **cluster-autoscaler** オブジェクトが存在する場合、**pods/eviction** ルールが動詞 **create** と共に存在することを確認します。

5. デプロイメント自動スケーラーのロールを作成します。

```
$ oc apply -n cluster-autoscaler -f - <<EOF
apiVersion: v1
kind: Role
metadata:
  name: cluster-autoscaler
rules:
- apiGroups:
  - ""
  resources:
  - configmaps
  resourceNames:
  - cluster-autoscaler
  - cluster-autoscaler-status
  verbs:
  - create
  - get
  - patch
  - update
  attributeRestrictions: null
- apiGroups:
  - ""
  resources:
  - configmaps
  verbs:
  - create
  attributeRestrictions: null
- apiGroups:
  - ""
  resources:
  - events
  verbs:
  - create
  attributeRestrictions: null
EOF
```

6. **creds** ファイルを作成して、自動スケーラーの AWS 認証情報を保存します。

```
$ cat <<EOF > creds
[default]
aws_access_key_id = your-aws-access-key-id
aws_secret_access_key = your-aws-secret-access-key
EOF
```

自動スケーラーはこれらの認証情報を使用して、新規インスタンスを起動します。

7. AWS 認証情報が含まれるシークレットを作成します。

```
$ oc create secret -n cluster-autoscaler generic autoscaler-credentials --from-file=creds
```

自動スケーラーはこのシークレットを使用して AWS 内でインスタンスを起動します。

8. cluster-reader ロールを作成し、これを作成した **cluster-autoscaler** サービスアカウントに付与します。

```
$ oc adm policy add-cluster-role-to-user cluster-autoscaler system:serviceaccount:cluster-autoscaler:cluster-autoscaler -n cluster-autoscaler

$ oc adm policy add-role-to-user cluster-autoscaler system:serviceaccount:cluster-autoscaler:cluster-autoscaler --role-namespace cluster-autoscaler -n cluster-autoscaler

$ oc adm policy add-cluster-role-to-user cluster-reader system:serviceaccount:cluster-autoscaler:cluster-autoscaler -n cluster-autoscaler
```

9. クラスターの自動スケーラーをデプロイします。

```
$ oc apply -n cluster-autoscaler -f - <<EOF
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: cluster-autoscaler
  name: cluster-autoscaler
  namespace: cluster-autoscaler
spec:
  replicas: 1
  selector:
    matchLabels:
      app: cluster-autoscaler
      role: infra
  template:
    metadata:
      labels:
app: cluster-autoscaler
role: infra
    spec:
      containers:
      - args:
- /bin/cluster-autoscaler
- --alsologtostderr
- --v=4
- --skip-nodes-with-local-storage=False
- --leader-elect-resource-lock=configmaps
- --namespace=cluster-autoscaler
- --cloud-provider=aws
- --nodes=0:6:mycluster-ASG
env:
- name: AWS_REGION
  value: us-east-1
- name: AWS_SHARED_CREDENTIALS_FILE
  value: /var/run/secrets/aws-creds/creds
image: registry.redhat.io/openshift3/ose-cluster-autoscaler:v3.11
name: autoscaler
```

```
volumeMounts:
- mountPath: /var/run/secrets/aws-creds
  name: aws-creds
  readOnly: true
    dnsPolicy: ClusterFirst
    nodeSelector:
node-role.kubernetes.io/infra: "true"
    serviceAccountName: cluster-autoscaler
    terminationGracePeriodSeconds: 30
    volumes:
    - name: aws-creds
secret:
  defaultMode: 420
  secretName: autoscaler-credentials
EOF
```

43.5. 自動スケーラーのテスト

自動スケーラーを Amazon Web Services (AWS) クラスターに追加した後に、現在のノードが実行できる数よりも多くの Pod をデプロイすることにより自動スケーラーが機能することを確認できます。

前提条件

- AWS で実行される OpenShift Container Platform クラスターに自動スケーラーが追加されていること。

手順

1. 自動スケーリングをテストするためのデプロイメント設定が含まれる **scale-up.yaml** ファイルを作成します。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: scale-up
  labels:
    app: scale-up
spec:
  replicas: 20 (1)
  selector:
    matchLabels:
      app: scale-up
  template:
    metadata:
      labels:
        app: scale-up
    spec:
      containers:
      - name: origin-base
        image: openshift/origin-base
        resources:
          requests:
            memory: 2Gi
        command:
        - /bin/sh
```

```
        - "-c"
        - "echo 'this should be in the logs' && sleep 86400"
      terminationGracePeriodSeconds: 0
```

1 このデプロイメントは 20 のレプリカを指定しますが、初期サイズのクラスターでは、最初にコンピュートノードの数を増やさないとすべての Pod を実行できません。

2. デプロイメントの namespace を作成します。

```
$ oc apply -f - <<EOF
apiVersion: v1
kind: Namespace
metadata:
  name: autoscaler-demo
EOF
```

3. 設定をデプロイします。

```
$ oc apply -n autoscaler-demo -f scale-up.yaml
```

4. namespace の Pod を表示します。

   a. namespace の実行中の Pod を表示します。

```
$ oc get pods -n autoscaler-demo | grep Running
cluster-autoscaler-5485644d46-ggvn5   1/1       Running   0          1d
scale-up-79684ff956-45sbg             1/1       Running   0          31s
scale-up-79684ff956-4kzjv             1/1       Running   0          31s
scale-up-79684ff956-859d2             1/1       Running   0          31s
scale-up-79684ff956-h47gv             1/1       Running   0          31s
scale-up-79684ff956-htjth             1/1       Running   0          31s
scale-up-79684ff956-m996k             1/1       Running   0          31s
scale-up-79684ff956-pvvrm             1/1       Running   0          31s
scale-up-79684ff956-qs9pp             1/1       Running   0          31s
scale-up-79684ff956-zwdpr             1/1       Running   0          31s
```

   b. namespace の保留中の Pod を表示します。

```
$ oc get pods -n autoscaler-demo | grep Pending
scale-up-79684ff956-5jdnj             0/1       Pending   0          40s
scale-up-79684ff956-794d6             0/1       Pending   0          40s
scale-up-79684ff956-7rlm2             0/1       Pending   0          40s
scale-up-79684ff956-9m2jc             0/1       Pending   0          40s
scale-up-79684ff956-9m5fn             0/1       Pending   0          40s
scale-up-79684ff956-fr62m             0/1       Pending   0          40s
scale-up-79684ff956-q255w             0/1       Pending   0          40s
scale-up-79684ff956-qc2cn             0/1       Pending   0          40s
scale-up-79684ff956-qjn7z             0/1       Pending   0          40s
scale-up-79684ff956-tdmqt             0/1       Pending   0          40s
scale-up-79684ff956-xnjhw             0/1       Pending   0          40s
```

   これらの保留中の Pod は、クラスターの自動スケーラーが Pod の実行に使用する新規コンピュートノードを自動的にプロビジョニングするまで実行できません。ノードがクラスター内で **Ready** 状態になるまで数分の時間がかかる場合があります。

5. 数分後に、新規ノードの準備ができているかどうかを確認するためにノードの一覧をチェックします。

```
$ oc get nodes
NAME                          STATUS    ROLES     AGE       VERSION
ip-172-31-49-172.ec2.internal   Ready     infra     1d        v1.11.0+d4cacc0
ip-172-31-53-217.ec2.internal   Ready     compute   7m        v1.11.0+d4cacc0
ip-172-31-55-89.ec2.internal    Ready     compute   9h        v1.11.0+d4cacc0
ip-172-31-56-21.ec2.internal    Ready     compute   7m        v1.11.0+d4cacc0
ip-172-31-56-71.ec2.internal    Ready     compute   7m        v1.11.0+d4cacc0
ip-172-31-63-234.ec2.internal   Ready     master    1d        v1.11.0+d4cacc0
```

6. 追加のノードの準備ができている場合、namespace で実行中の Pod を再度表示します。

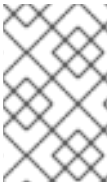
```
$ oc get pods -n autoscaler-demo
NAME                                READY     STATUS    RESTARTS   AGE
cluster-autoscaler-5485644d46-ggvn5   1/1       Running   0          1d
scale-up-79684ff956-45sbg             1/1       Running   0          8m
scale-up-79684ff956-4kzjv             1/1       Running   0          8m
scale-up-79684ff956-5jdnj             1/1       Running   0          8m
scale-up-79684ff956-794d6             1/1       Running   0          8m
scale-up-79684ff956-7rlm2             1/1       Running   0          8m
scale-up-79684ff956-859d2             1/1       Running   0          8m
scale-up-79684ff956-9m2jc             1/1       Running   0          8m
scale-up-79684ff956-9m5fn             1/1       Running   0          8m
scale-up-79684ff956-fr62m             1/1       Running   0          8m
scale-up-79684ff956-h47gv             1/1       Running   0          8m
scale-up-79684ff956-htjth             1/1       Running   0          8m
scale-up-79684ff956-m996k             1/1       Running   0          8m
scale-up-79684ff956-pvvrm             1/1       Running   0          8m
scale-up-79684ff956-q255w             1/1       Running   0          8m
scale-up-79684ff956-qc2cn             1/1       Running   0          8m
scale-up-79684ff956-qjn7z             1/1       Running   0          8m
scale-up-79684ff956-qs9pp             1/1       Running   0          8m
scale-up-79684ff956-tdmqt             1/1       Running   0          8m
scale-up-79684ff956-xnjhw             1/1       Running   0          8m
scale-up-79684ff956-zwdpr             1/1       Running   0          8m
...
```

第44章 機能ゲートの使用による各種機能の無効化

管理者として、機能ゲート を使用して特定のノードまたはプラットフォーム全体に対して特定の機能をオフにすることができます。

たとえば、新規の機能を完全にテストできるテストクラスターで有効にした状態で、これらの機能を実稼働クラスターでオフにすることができます。

Web コンソールに表示される機能を無効にすると、その機能が表示される場合がありますが、オブジェクトは一覧表示されません。無効にされた機能に関連付けられたコマンドを使用しようとすると、OpenShift Container Platform にエラーが表示されます。



注記

クラスター内のアプリケーションが依存する機能を無効にする場合、アプリケーションは無効にされた機能およびアプリケーションがその機能を使用する方法によっては適切に機能しなくなる場合があります。

機能ゲートは、ブロックする必要のある機能を記述したマスター設定ファイル (**/etc/origin/master/master-config.yaml**) およびノード設定ファイルで **key=value** のペアを使用します。

ノード設定ファイルを変更するには、ノード設定マップ を必要に応じて更新します。**node-config.yaml** ファイルは手動で変更しないようにしてください。

たとえば、以下のコードは Huge Page 機能をオフにします。

```
kubernetesMasterConfig:
  apiServerArguments:
    feature-gates:
    - HugePages=false (1)
  ...
  controllerArguments:
    feature-gates:
    - HugePages=false (2)
```

(1) (2) 機能をオフにするキー/値のペア:

- **true**: 指定された機能を有効にします。
- **false**: 指定された機能を無効にします。

複数の機能ゲートを 1 つのコンマ区切り行で指定します。

```
kubeletArguments:
  feature-gates:
  -
RotateKubeletClientCertificate=true,RotateKubeletServerCertificate=true,ExpandPersistentVolumes=true,HugePages=false
```

44.1. クラスターの各種機能の無効化

クラスター全体の機能をオフにするには、マスター設定ファイル (デフォルトは **/etc/origin/master/master-config.yaml**) を編集します。

1. オフにする必要のある機能については、**<feature_name>=false** を **apiServerArguments** および **controllerArguments** の下に入力します。
以下に例を示します。

```
kubernetesMasterConfig:
  apiServerArguments:
    feature-gates:
    - HugePages=false
  controllerArguments:
    feature-gates:
    - HugePages=false
```

複数の機能ゲートを 1 つのコンマ区切り行で指定します。

```
kubernetesMasterConfig:
  apiServerArguments:
    feature-gates:
    -
RotateKubeletClientCertificate=false,RotateKubeletServerCertificate=false,ExpandPersistent
Volumes=true,HugePages=false
  controllerArguments:
    feature-gates:
    -
RotateKubeletClientCertificate=false,RotateKubeletServerCertificate=false,ExpandPersistent
Volumes=true,HugePages=false
```

2. 変更を有効にするために OpenShift Container Platform マスターサービスを再起動します。

```
# master-restart api
# master-restart controllers
```

無効にされた機能を再度有効にするには、マスター設定ファイルを編集して **<feature_name>=false** を削除し、マスターサービスを再起動します。

44.2. ノードの各種機能の無効化

ノードホストの機能をオフにするには、適切な ノード設定マップ を編集します。

ノード設定ファイルを変更するには、ノード設定マップ を必要に応じて更新します。**node-config.yaml** ファイルは手動で変更しないようにしてください。

1. オフにする必要のある機能については、**<feature_name>=false** を **kubeletArguments** の下に入力します。
以下に例を示します。

```
kubeletArguments:
  feature-gates:
  - HugePages=false
```

複数の機能ゲートを 1 つのコンマ区切り行で指定します。

```
kubeletArguments:
  feature-gates:
  -
RotateKubeletClientCertificate=false,RotateKubeletServerCertificate=false,ExpandPersistent
Volumes=true,HugePages=false
```

2. 変更を有効にするために OpenShift Container Platform サービスを再起動します。

```
# systemctl restart atomic-openshift-node.service
```

無効にされた機能を再度有効にするには、ノード設定ファイルを編集して **<feature_name>=false** を削除し、ノードサービスを再起動します。

ノード設定ファイルを変更するには、ノード設定マップ を必要に応じて更新します。**node-config.yaml** ファイルは手動で変更しないようにしてください。

44.2.1. 機能ゲートの一覧

以下の一覧を使用し、無効にする必要のある機能の名前を判別します。

機能ゲート	説明
Accelerator	Docker の使用時に Nvidia GPU サポートを有効にします。
AdvancedAuditing	高度な監査 を有効にします。
APIListChunking	API クライアントが LIST または GET リソースを API サーバーからチャンクで取得できるようにします。
APIResponseCompression	LIST または GET 要求の応答の圧縮を有効にします。
AppArmor	Docker を使用する際に Linux ノードで AppArmor ベースの必須 no アクセス制御を有効にします。詳細は、Kubernetes AppArmor ドキュメント を参照してください。
BlockVolume	Pod で raw ブロックデバイスの定義および消費を可能にします。詳細は、Kubernetes Raw Block Volume Support を参照してください。
CPUManager	コンテナーレベルの CPU アフィニティーサポートを有効にします。詳細は、Using CPU Manager を参照してください。
CRIContainerLogRotation	CRI コンテナーランタイムのコンテナーログのローテーションを有効にします。

機能ゲート	説明
CSIPersistentVolume	CSI (Container Storage Interface) と互換性のあるボリュームプラグインでプロビジョニングされるボリュームの検出およびモニターを有効にします。詳細は、CSI Volume Plugins(Kubernetes Design ドキュメント) を参照してください。
CustomPodDNS	**dnsConfig** プロパティーを使用した Pod の DNS 設定のカスタマイズを有効にします。
CustomResourceSubresources	CustomResourceDefinition で作成されたリソースで **/status** および **/scale** サブリソースを有効にします。カスタムリソースによる Kubernetes API の拡張を参照してください。
CustomResourceValidation	カスタムリソース定義で作成されたリソースでスキーマベースの検証を有効にします。詳細は、カスタムリソースによる Kubernetes API の拡張 を参照してください。
DebugContainers	実行中の Pod のトラブルシューティングのために Pod の namespace でのデバッグコンテナーの実行を有効にします。
DevicePlugins	ノード上で デバイスプラグインベース のリソースのプロビジョニングを有効にします。
DynamicKubeletConfig	クラスター内で動的な設定を有効にします。
DynamicVolumeProvisioning(deprecated)	永続ボリュームの Pod への動的プロビジョニングを有効にします。
EnableEquivalenceClassCache	Pod のスケジュール時のスケジューラーによるノードの同等内容のキャッシュを有効にします。
ExperimentalCriticalPodAnnotation	特定の Pod に Critical のアノテーションを付け、それらのスケジューリングが保証されるようにします。
ExperimentalHostUserNamespaceDefaultingGate	ユーザー namespace の無効化を可能にします。これは、権限の付けられた他のホストプロジェクト、ホストマウント、またはコンテナーを使用するか、または MKNODE、SYS_MODULE などの特定のプロジェクト以外の機能を使用するコンテナー用です。これは、ユーザープロジェクトの再マップが Docker デーモンで有効にされる場合にのみ有効にされる必要があります。
GCERegionalPersistentDisk	GCE 永続ディスク 機能を有効にします。

機能ゲート	説明
HugePages	事前に割り当てられた Huge Page の割り当ておよび消費を有効にします。
HyperVContainer	Windows コンテナーの Hyper-V の分離を有効にします。
Intializers	ビルトインの受付コントローラーの拡張として 動的な受付制御 を有効にします。
LocalStorageCapacityIsolation	ローカルの一時ストレージの消費、および **emptyDir** ボリュームの **sizeLimit** プロパティーを有効にします。
MountContainers	ホストでのユーティリティーコンテナーのボリュームマウントとしての使用を有効にします。
MountPropagation	1 つのコンテナーでマウントされたボリュームの他のコンテナーまたは Pod との共有を有効にします。
PersistentLocalVolumes	ローカルボリューム Pod の使用を有効にします。Pod のアフィニティーは、ローカルボリュームを要求する場合に指定される必要があります。
PodPriority	優先順位に基づく Pod の 再スケジュール (Descheduling) およびプリエンプションを有効にします。
ReadOnlyAPIDataVolumes	シークレット、ConfigMap、DownwardAPI、および Projected ボリューム が読み取り専用モードでマウントされるように設定します。
ResourceLimitsPriorityFunction	最も低いスコアの **1** を、1 つ違法の入力 Pod CPU およびメモリー制限を満たすノードを割り当てる scheduler の優先度関数を有効にします。これは、同じスコアを持つノード間の関連性を切り離すことを目的としています。
RotateKubeletClientCertificate	クラスター上でのクラスター TLS 証明書のローテーションを有効にします。
RotateKubeletServerCertificate	クラスター上でのサーバー TLS 証明書のローテーションを有効にします。
RunAsGroup	コンテナーの init プロセスのプライマリーグループ ID セットに対する制御を有効にします。

機能ゲート	説明
ScheduleDaemonSetPods	DaemonSet Pod を DaemonSet コントローラーではなく デフォルトスケジューラー でスケジュールされることを可能にします。
ServiceNodeExclusion	クラウドプロバイダーで作成されたロードバランサーからのノードの除外を有効にします。
StorageObjectInUseProtection	永続ボリューム または Persistent Volume Claim (永続ボリューム要求、PVC) オブジェクトが依然として使用されている場合、これらの削除の延期を有効にします。
StreamingProxyRedirects	API サーバーに対し、ストリーミング要求についてインターセプトおよびバックエンド kubelet からのリダイレクトをフォローするように指示します。
SupportIPVSProxyMode	IP 仮想サーバーを使用したクラスター内のサービスの負荷分散を有効にします。
SupportPodPidsLimit	Pod で実行されるプロセス数 (PID) を制限するサポートを有効にします。
TaintBasedEvictions	ノードのテイントおよび Pod の容認 に基づく Pod のノードからのエビクトを有効にします。
TaintNodesByCondition	ノードの状態に基づく自動的な ノードのテイント を有効にします。
TokenRequest	サービスアカウントリソースで TokenRequest エンドポイントを有効にします。
VolumeScheduling	ボリュームトポロジー対応のスケジューリングを有効にし、Persistent Volume Claim (永続ボリューム要求、PVC) のバインディングにスケジューリングの意思決定を認識させます。また、**PersistentLocalVolumes** 機能ゲートと使用される場合の local volumes タイプの使用も有効にします。

第45章 KURYR SDN の管理

45.1. 概要

Kuryr (または Kuryr-Kubernetes) は OpenShift Container Platform における SDN のオプションの 1 つです。Kuryr は OpenStack ネットワークサービスの Neutron を使用して Pod をネットワークに接続します。この方法で、Pod は OpenStack 仮想マシン (VM) との内部の接続を確保します。 これは、OpenStack VM でデプロイされた OpenShift Container Platform クラスターに役立ちます。

45.1.1. 孤立した OpenStack リソース

Kuryr で作成されたすべての OpenStack リソースは、OpenShift Container Platform リソースのライフサイクルに関連付けられます。OpenStack VM などの Kuryr で作成されたリソースを手動で削除すると、OpenStack デプロイメントに孤立したリソースが生じる可能性があります。孤立したリソースは、**kuryr.conf** ファイル内にあるリソース ID を検索して適切に削除できます。または、Kuryr リソースが別の OpenStack ユーザーによって作成されている場合、ユーザーは関連付けられたユーザー名で OpenStack API をクエリーできます。