OpenShift Container Platform 3.9

クラスター管理

OpenShift Container Platform 3.9 Cluster Administration

Last Updated: 2021-05-10

OpenShift Container Platform 3.9 クラスター管理

OpenShift Container Platform 3.9 Cluster Administration

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 MANAGING NODES

2.1. 概要

You can manage nodes in your instance using the CLI.

When you perform node management operations, the CLI interacts with node objects that are representations of actual node hosts. The master uses the information from node objects to validate nodes with health checks.

2.2. LISTING NODES

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
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
```

To only list information about a single node, replace **<node>** with the full node name:

```
$ 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。
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ついての詳細情報を取得するには、以下を実行します。

```
$ oc describe node <node>
```

例:

```
$ oc describe node node1.example.com
Name:   node1.example.com
```

```
Labels:   kubernetes.io/hostname=node1.example.com
CreationTimestamp: Wed, 10 Jun 2015 17:22:34 +0000
Conditions:
  Type  Status LastHeartbeatTime   LastTransitionTime   Reason     Message
  Ready  True  Wed, 10 Jun 2015 19:56:16 +0000  Wed, 10 Jun 2015 17:22:34 +0000  kubelet is
posting ready status
Addresses: 127.0.0.1
Capacity:
 memory: 1017552Ki
 pods:  100
 cpu:  2
Version:
 Kernel Version:  3.17.4-301.fc21.x86_64
 OS Image:   Fedora 21 (Twenty One)
 Container Runtime Version: docker://1.6.0
 Kubelet Version:  v0.17.1-804-g496be63
 Kube-Proxy Version:  v0.17.1-804-g496be63
ExternalID:   node1.example.com
Pods:    (2 in total)
  docker-registry-1-9yyw5
  router-1-maytv
No events.
```

2.3. ADDING NODES

To add nodes to your existing OpenShift Container Platform cluster, you can run an Ansible playbook that handles installing the node components, generating the required certificates, and other important steps. See the advanced installation method for instructions on running the playbook directly.

Alternatively, if you used the quick installation method, you can re-run the installer to add nodes , which performs the same steps.

2.4. DELETING NODES

When you delete a node using the CLI, the node object is deleted in Kubernetes, but the pods that exist on the node itself are not deleted. Any bare pods not backed by a replication controller would be inaccessible to OpenShift Container Platform, pods backed by replication controllers would be rescheduled to other available nodes, and local manifest pods would need to be manually deleted.

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ます。

4. If you want to uninstall all OpenShift Container Platform content from the node host, including

all pods and containers, continue to Uninstalling Nodes and follow the procedure using the **uninstall.yml** playbook. The procedure assumes general understanding of the advanced installation method using Ansible.

2.5. UPDATING LABELS ON NODES

To add or update labels on a node:

```
$ 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
```

2.6. LISTING PODS ON NODES

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]
```

選択したノードの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]
```

2.7. MARKING NODES AS UNSCHEDULABLE OR SCHEDULABLE

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
```

例:

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

2.8. EVACUATING PODS ON NODES

Evacuating pods allows you to migrate all or selected pods from a given node or nodes. Nodes must first be marked unschedulable to perform pod evacuation.

Only pods backed by a replication controller can be evacuated; the replication controllers create new pods on other nodes and remove the existing pods from the specified node(s). Bare pods, meaning those not backed by a replication controller, are unaffected by default.

To evacuate all or selected pods on one or more nodes:

```
$ oc adm drain <node1> <node2> [--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、daemonset、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 <node1> <node2> --force=true
```

You can use **--grace-period** to set a period of time in seconds for each pod to terminate gracefully. If negative, the default value specified in the pod will be used:

```
$ oc adm drain <node1> <node2> --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 <node1> <node2> --ignore-daemonset=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 <node1> <node2> --timeout=5s
```

You can use **--delete-local-data** and set it to **true** to continue deletion even if there are pods using emptyDir (local data that will be deleted when the node is drained):

```
$ oc adm drain <node1> <node2> --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 <node1> <node2>  --dry-run=true
```

Instead of specifying specific node names (for example, **<node1> <node2>**), you can use the **--selector=<node_selector>** option to evacuate pods on selected nodes.

2.9. REBOOTING NODES

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。

2.9.1. Infrastructure Nodes

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。最善の解決法として、Pod の非アフィニティーを使用できます。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。

2.9.2. Using Pod Anti-affinity

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。

Using the **docker-registry** pod as an example, the first step in enabling this feature is to set the **scheduler.alpha.kubernetes.io/affinity** on the pod. Since this pod uses a deployment configuration, the most appropriate place to add the annotation is to the pod template's metadata.

```
$ oc edit dc/docker-registry -o yaml

...
  template:
    metadata:
      annotations:
        scheduler.alpha.kubernetes.io/affinity: |
          {
            "podAntiAffinity": {
              "requiredDuringSchedulingIgnoredDuringExecution": [{
                "labelSelector": {
                  "matchExpressions": [{
```

```
            "key": "docker-registry",
            "operator": "In",
            "values":["default"]
          }]
        },
        "topologyKey": "kubernetes.io/hostname"
      }]
    }
  }
```

重要

scheduler.alpha.kubernetes.io/affinity is internally stored as a string even though the contents are JSON. The above example shows how this string can be added as an annotation to a YAML deployment configuration.

This example assumes the Docker registry pod has a label of **docker-registry=default**. Pod anti-affinity can use any Kubernetes match expression.

The last required step is to enable the **MatchInterPodAffinity** scheduler predicate in **/etc/origin/master/scheduler.json**. With this in place, if only two infrastructure nodes are available and one is rebooted, the Docker registry pod is prevented from running on the other node. **oc get pods** reports the pod as unready until a suitable node is available. Once a node is available and all pods are back in ready state, the next node can be restarted.

2.9.3. Handling Nodes Running Routers

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。

In rare cases, a router pod may not have a host port configured. In those cases, it is important to follow the recommended restart process for infrastructure nodes.

2.10. ノードリソースの設定

You can configure node resources by adding kubelet arguments to the node configuration file (**/etc/origin/node/node-config.yaml**). Add the **kubeletArguments** section and include any desired options:

```
kubeletArguments:
  max-pods: (1)
    - "40"
  resolv-conf: (2)
    - "/etc/resolv.conf"
  image-gc-high-threshold: (3)
    - "90"
  image-gc-low-threshold: (4)
    - "80"
```

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。デフォルト: 80%

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。

```
$ kubelet -h
```

This can also be set during an advanced installation using the **openshift_node_kubelet_args** variable. For example:

```
openshift_node_kubelet_args={'max-pods': ['40'], 'resolv-conf': ['/etc/resolv.conf'],  'image-gc-high-threshold': ['90'], 'image-gc-low-threshold': ['80']}
```

2.10.1. Setting Maximum Pods Per Node

注記

See the Cluster Limits page for the maximum supported limits for each version of OpenShift Container Platform.

/etc/origin/node/node-config.yaml ファイルでは、 **pods-per-core** および **max-pods** の 2 つのパラメーターがノードにスケジュールできる Pod の最大数を制御します。いずれのオプションも使用されている場合、2 つの内の小さい方の値でノードの Pod 数が制限されます。これらの値を超えると、以下の状況が発生します。

- 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。

pods-per-core は、ノードのプロセッサーコア数に基づいてノードが実行できる Pod 数を設定します。たとえば、4 プロセッサーコアを搭載したノードで **pods-per-core** が **10** に設定される場合、このノードで許可される Pod の最大数は 40 になります。

■

```
kubeletArguments:
  pods-per-core:
    - "10"
```

注記

pods-per-core を 0 に設定すると、この制限が無効になります。

max-pods sets the number of pods the node can run to a fixed value, regardless of the properties of the node. Cluster Limits documents maximum supported values for **max-pods**.

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

上記の例では、**pods-per-core** のデフォルト値は **10** であり、**max-pods** のデフォルト値は **250** です。これは、ノードにあるコア数が 25 以上でない限り、デフォルトでは **pods-per-core** が制限を設定することになります。

2.11. RESETTING DOCKER STORAGE

As you download Docker images and run and delete containers, Docker does not always free up mapped disk space. As a result, over time you can run out of space on a node, which might prevent OpenShift Container Platform from being able to create new pods or cause pod creation to take several minutes.

For example, the following shows pods that are still in the **ContainerCreating** state after six minutes and the events log shows a FailedSync event.

```
$ oc get pod
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
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 ストレージを再設定します。

```
$ 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. 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
```

2.12. CHANGING NODE TRAFFIC INTERFACE

By default, DNS routes all node traffic. During node registration, the master receives the node IP addresses from the DNS configuration, and therefore accessing nodes via DNS is the most flexible solution for most deployments.

If your deployment is using a cloud provider, then the node gets the IP information from the cloud provider. However, **openshift-sdn** attempts to determine the IP through a variety of methods, including a DNS lookup on the nodeName (if set), or on the system hostname (if nodeName is not set).

However, you may need to change the node traffic interface. For example, where:

- OpenShift Container Platform is installed in a cloud provider where internal hostnames are not configured/resolvable by all hosts.
- The node's IP from the master's perspective is not the same as the node's IP from its own perspective.

Configuring the **openshift_set_node_ip** Ansible variable forces node traffic through an interface other than the default network interface.

To change the node traffic interface:

1. Set the **openshift_set_node_ip** Ansible variable to **true**.
2. Set the **openshift_ip** to the IP address for the node you want to configure.

注記

Although **openshift_set_node_ip** can be useful as a workaround for the cases stated in this section, it is generally not suited for production environments. This is because the node will no longer function properly if it receives a new IP address.

第3章 ユーザーの管理

3.1. 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。

This topic describes the management of user accounts, including how new user accounts are created in OpenShift Container Platform and how they can be deleted.

3.2. ユーザーの作成

The process for creating a user depends on the configured identity provider. By default, OpenShift Container Platform uses the **DenyAll** identity provider, which denies access for all user names and passwords.

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. Create the user account depending on your identity provider. This can depend on the **mappingmethod** used as part of the identity provider configuration. See the Mapping Identities to Users section for more information.
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
```

For an explanation and list of roles, see the Cluster Roles and Local Roles section of the Architecture Guide.

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

Depending on the identity provider, and on the defined group structure, some roles may be given to users automatically. See the Synching groups with LDAP section for more information.

3.3. 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。

3.4. 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構成される 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。

If your organization is using LDAP, you can synchronize any LDAP records to OpenShift Container Platform so that you can configure groups on one place. This presumes that information about your users is in an MDAP server. See the Synching groups with LDAP section for more information. If you are not using LDAP, you can use the following procedure to manually create groups.

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
```

Next steps: * Managing role bindings

3.5. 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>
```

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>
```

3.6. 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4章 プロジェクトの管理

4.1. 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。

4.2. プロジェクトのセルフプロビジョニング

You can allow developers to create their own projects. There is an endpoint that will provision a project according to a template. The web console and **oc new-project** command use this endpoint when a developer creates a new project .

4.2.1. 新規プロジェクトのテンプレートの変更

The API server automatically provisions projects based on the template that is identified by the **projectRequestTemplate** parameter of the **master-config.yaml** file. If the parameter is not defined, the API server creates a default template that creates a project with the requested name, and assigns the requesting user to the "admin" role for that project.

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で **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。

Access to the API is granted to developers with the **self-provisioner** role and the **self-provisioners** cluster role binding. This role is available to all authenticated developers by default.

4.2.2. セルフプロビジョニングの無効化

Removing the **self-provisioners**cluster role from authenticated user groups will deny permissions for self-provisioning any new projects.

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

When disabling self-provisioning, set the **projectRequestMessage** parameter in the **master-config.yaml** file to instruct developers on how to request a new project. This parameter is a string that will be presented to the developer in the web console and command line when they attempt to self-provision a project. For example:

```
Contact your system administrator at projectname@example.com to request a project.
```

or:

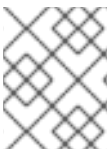
```
To request a new project, fill out the project request form located at
https://internal.example.com/openshift-project-request.
```

サンプル YAML ファイル

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

4.3. 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

Labels can be assigned during an advanced installation, or added to a node after installation.

4.3.1. 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。

```
# systemctl restart atomic-openshift-master-api atomic-openshift-master-controllers
```

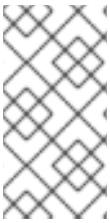
4.3.2. プロジェクト全体でのノードセレクターの設定

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'
```

いったん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":"region=infra"}}}'
```

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。

4.3.3. 開発者が指定するノードセレクター

OpenShift Container Platform developers can set a node selector on their pod configuration if they wish to restrict nodes even further. This will be in addition to the project node selector, meaning that you can still dictate node selector values for all projects that have a node selector value.

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。

4.4. 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

The number of self-provisioned projects requested by a given user can be limited with the **ProjectRequestLimit**admission control plug-in.

重要

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 OpenShift Container Platform 3.1 (またはそれ以前のバージョン) で作成される場合、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。

In order to specify limits for users, a configuration must be specified for the plug-in within the master configuration file (**/etc/origin/master/master-config.yaml**). The plug-in configuration takes a list of user label selectors and the associated maximum project requests.

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、ラベル **level=admin** を持つユーザーに対して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。

```
# systemctl restart atomic-openshift-master-api atomic-openshift-master-controllers
```

第5章 POD の管理

5.1. 概要

This topic describes the management of pods, including limiting their run-once duration, and how much bandwidth they can use.

5.2. 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。

5.2.1. 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、プロジェクト別の期限によって置き換えられることがあります。

```
kubernetesMasterConfig:
  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。

5.2.2. 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。

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

(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。

5.2.2.1. Egress ルーター Pod のデプロイ

例5.1 Egress ルーターの Pod 定義のサンプル

```
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ースに Macvlan ネットワークインターフェースを作成してから、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

Preserve the the quotation marks around **"true"**. Omitting them will result in errors.

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 インターフェースを設定したり、**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。

5.2.2.2. Egress ルーターサービスのデプロイ

通常、egress ルーターを参照するサービスを作成する必要が生じる場合があります (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。

5.2.3. 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

You must have the **ovs-multitenant** plug-in enabled in order to limit pod access via egress policy.

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。

5.2.3.1. 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、オブジェクトの機能方法についての 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 の制限ポリシーについての 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。以下は例になります。

```
{
   "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** へのへのトラフィックは許可されますが、その他すべての外部 IP アドレスへのアクセスは拒否されます (ポリシーが **外部** トラフィックにのみ適用されるので他の 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。

5.3. 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。

例5.2 制限が設定された Pod オブジェクト定義

```
{
    "kind": "Pod",
    "spec": {
        "containers": [
            {
                "image": "nginx",
                "name": "nginx"
            }
        ]
    },
    "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>
```

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

A **pod disruption budget**is part of the Kubernetes API, which can be managed with **oc** commands like other object types. They allow the specification of safety constraints on pods during operations, such as draining a node for maintenance.

注記

Starting in OpenShift Container Platform 3.6, pod disruption budgets are now fully supported.

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

PodDisruptionBudget オブジェクトの設定は、以下の主要な部分で構成されています。

- 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** の 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。

5.5. INJECTING INFORMATION INTO PODS USING POD PRESETS

A **pod preset** is an object that injects user-specified information into pods as they are created.

重要

Pod presets is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs), might not be functionally complete, and Red Hat does not recommend to use them for production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

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。

Using pod preset objects you can inject:

- secret objects
- **ConfigMap** objects
- storage volumes
- container volume mounts
- environment variables

Developers only need make sure the pod labels match the label selector on the PodPreset in order to add all that information to the pod. The label on a pod associates the pod with one or more pod preset objects that have a matching label selectors.

Using pod presets, a developer can provision pods without needing to know the details about the services the pod will consume. An administrator can keep configuration items of a service invisible from a developer without preventing the developer from deploying pods. For example, an administrator can create a pod preset that provides the name, user name, and password for a database through a secret and the database port through environment variables. The pod developer only needs to know the label to use to include all the information in pods. A developer can also create pod presets and perform all the same tasks. For example, the developer can create a preset that injects environment variable automatically into multiple pods.

注記

The Pod Preset feature is available only if the Service Catalog has been installed.

You can exclude specific pods from being injected using the **podpreset.admission.kubernetes.io/exclude: "true"** parameter in the pod specification. See the example pod specification.

For more information, see Injecting Information into Pods Using Pod Presets .

第6章 ネットワークの管理

6.1. 概要

This topic describes the management of the overall cluster network, including project isolation and outbound traffic control.

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。

6.2. POD ネットワークの管理

When your cluster is configured to use the **ovs-multitenant** SDN plugin, you can manage the separate pod overlay networks for projects using the administrator CLI. See the Configuring the SDN section for plug-in configuration steps, if necessary.

6.2.1. 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>
```

In the above example, all the pods and services in **<project2>** and **<project3>** can now access any pods and services in **<project1>** and vice versa. Services can be accessed either by IP or fully-qualified DNS name (**<service>.<pod_namespace>.svc.cluster.local**). For example, to access a service named **db** in a project **myproject**, use **db.myproject.svc.cluster.local**.

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。

6.3. 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。

6.3.1. 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、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。

6.4. 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。

```
$ systemctl restart atomic-openshift-master-api atomic-openshift-master-controllers
```

6.5. 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。

The egress pod's deployment declares one of the source IPs, the destination IP of the protected service, and a gateway IP to reach the destination. After the pod is deployed, you can create a service to access the egress router pod, then add that source IP to the corporate firewall. The developer then has access information to the egress router service that was created in their project, for example, **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。

You can also assign static IP addresses to projects, ensuring that all outgoing external connections from the specified project have recognizable origins. This is different from the default egress router, which is used to send traffic to specific destinations. See the Enabling Fixed IPs for External Project Traffic section for more information.

注記

The egress router is not available for OpenShift Dedicated.

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。

6.5.1. 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

As an OpenShift Container Platform cluster administrator, you can use egress firewall policy to limit the external addresses that some or all pods can access from within the cluster, so that:

- 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。

You can configure projects to have different egress policies. For example, allowing **<project A>** access to a specified IP range, but denying the same access to **<project B>**. Or restrict application developers from updating from (Python) pip mirrors, and forcing updates to only come from desired sources.

注意

You must have the **ovs-multitenant** or **ovs-networkpolicy** plug-in enabled in order to limit pod access via egress policy.

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]** を使用して **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. Create a JSON file with the desired policy details. For example:

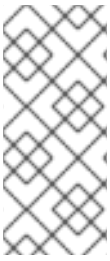
```
{
    "kind": "EgressNetworkPolicy",
    "apiVersion": "v1",
    "metadata": {
        "name": "default"
    },
    "spec": {
        "egress": [
            {
                "type": "Allow",
                "to": {
                    "cidrSelector": "1.2.3.0/24"
                }
            },
```

```
        {
            "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。

Domain name updates are polled based on the TTL (time to live) value of the domain of the local non-authoritative server, or 30 minutes if the TTL is unable to be fetched. The pod should also resolve the domain from the same local non-authoritative server when necessary, otherwise the IP addresses for the domain perceived by the egress network policy controller and the pod will be different, and the egress network policy may not be enforced as expected. In the above example, suppose **www.foo.com** resolved to **10.11.12.13** and has a DNS TTL of one minute, but was later changed to **20.21.22.23**. OpenShift Container Platform will then take up to one minute to adapt to these changes.



注記

The egress firewall always allows pods access to the external interface of the node the pod is on for DNS resolution. If your DNS resolution is not handled by something on the local node, then you will need to add egress firewall rules allowing access to the DNS server's IP addresses if you are using domain names in your pods. The default installer sets up a local dnsmasq, so if you are using that setup you will not need to add extra rules.

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

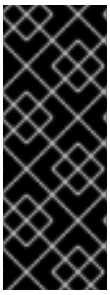
```
$ oc create -f <policy>.json
```

注意

Exposing services by creating routes will ignore **EgressNetworkPolicy**. Egress network policy service endpoint filtering is done at the node **kubeproxy**. When the router is involved, **kubeproxy** is bypassed and egress network policy enforcement is not applied. Administrators can prevent this bypass by limiting access to create routes.

6.5.2. 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ー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

Currently, the egress router is not compatible with Amazon AWS due to AWS not being compatible with macvlan traffic.

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ースに追加します。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 を使って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。これを行わないと、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、 **EnableMACAntiSpoofingFilterRules** を **false** に設定している必要があります。

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

The egress router can run in two different modes: redirect mode and HTTP proxy mode. Redirect mode works for all services except for HTTP and HTTPS. For HTTP and HTTPS services, use HTTP proxy mode.

6.5.2.1. 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. 上記を使用して 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.access.redhat.com/openshift3/ose-egress-router
    securityContext:
      privileged: true
    env:
    - name: EGRESS_SOURCE (2)
      value: 192.168.12.99
    - 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.access.redhat.com/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ースで Macvlan ネットワークインターフェースを作成し、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"** の周りの引用符はそのまま残します。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ース以外のネットワークインターフェースで Macvlan インターフェースを作成するには、アノテーションの値を該当インターフェースの名前に設定します。たとえば、 **eth1** を使用します。

(2) IP address from the physical network that the node is on and is reserved by the cluster administrator for use by this pod.

(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 インターフェースを設定し、**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。

6.5.2.2. 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。ポートによっては複数の異なる宛先 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.access.redhat.com/openshift3/ose-egress-router
    securityContext:
      privileged: true
    env:
    - name: EGRESS_SOURCE
      value: 192.168.12.99
    - name: EGRESS_GATEWAY
      value: 192.168.12.1
    - name: EGRESS_DESTINATION
      value: | (1)
        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.access.redhat.com/openshift3/ose-pod
```

(1) This uses the YAML syntax for a multi-line string; see below for details.

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>** - This says that incoming connections to the given **<port>** should be redirected to the same port on the given **<IP address>**. **<protocol>** is either **tcp** or **udp**. In the example above, the first line redirects traffic from local port 80 to port 80 on 203.0.113.25.

- **<port> <protocol> <IP address> <remote port>** - As above, except that the connection is redirected to a different **<remote port>** on **<IP address>**. In the example above, the second and third lines redirect local ports 8080 and 8443 to remote ports 80 and 443 on 203.0.113.26.

- **<fallback IP address>** - If the last line of **EGRESS_DESTINATION** is a single IP address, then any connections on any other port will be redirected to the corresponding port on that IP address (eg, 203.0.113.27 in the example above). If there is no fallback IP address then connections on other ports would simply be rejected.)

6.5.2.3. 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
  value: 192.168.12.99
- name: EGRESS_GATEWAY
  value: 192.168.12.1
- name: EGRESS_DESTINATION
  valueFrom:
    configMapKeyRef:
      name: egress-routes
      key: destination
- name: EGRESS_ROUTER_MODE
  value: init
...
```

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。

6.5.2.4. 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.access.redhat.com/openshift3/ose-egress-router
    securityContext:
      privileged: true
    env:
    - name: EGRESS_SOURCE (2)
      value: 192.168.12.99
    - name: EGRESS_GATEWAY (3)
      value: 192.168.12.1
    - name: EGRESS_ROUTER_MODE (4)
      value: http-proxy
  containers:
  - name: egress-router-proxy
    image: registry.access.redhat.com/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ースで Macvlan ネットワークインターフェースを作成してから、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) An IP address from the physical network that the node itself is on and is reserved by the cluster administrator for use by this pod.

(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。

6.5.2.5. 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.access.redhat.com/openshift3/ose-egress-router
        env:
        - name: EGRESS_SOURCE
          value: 192.168.12.99
        - 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.access.redhat.com/openshift3/ose-pod
      nodeSelector:
        site: springfield-1
```

(1) **replicas** が **1** に設定されていることを確認します。1 つの Pod のみが指定される **EGRESS_SOURCE** 値を随時使用できるためです。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. この定義を使用して 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>
```

6.5.3. 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。

6.6. 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、トラフィックを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 ルーターの場合とは異なります。

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。

静的 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"]}'
```

The **egressIPs** field is an array, but must be set to a single IP address. If setting multiple IPs, the other IPs will be ignored.

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 IPs 192.168.1.100、192.168.1.101 および 192.168.1.102 が必要である場合が、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 IPs are implemented as additional IP addresses on the primary network interface, and must be in the same subnet as the node's primary IP. Allowing additional IP addresses on the primary network interface might require extra configuration when using some cloud or VM solutions.

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。

6.7. 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。

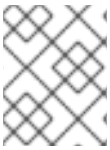
Multicast traffic between OpenShift Container Platform pods is disabled by default. You can enable Multicast on a per-project basis by setting an annotation on the project's corresponding **netnamespace** object:

```
$ 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-
```

If you have joined networks together, you will need to enable Multicast in each projects' **netnamespace** in order for it to take effect in any of the projects. To enable Multicast in the **default** project, you must also enable it in the **kube-service-catalog** project and all other projects that have been made global.



注記

Multicast global projects are not "global", but instead communicate with only other global projects via Multicast, not with all projects in the cluster, as is the case with unicast.

6.8. NETWORKPOLICY の有効化

The **ovs-subnet** and **ovs-multitenant** plugins have their own legacy models of network isolation, and don't support Kubernetes **NetworkPolicy**. However, **NetworkPolicy** support is available by using the **ovs-networkpolicy** plug-in.

In a cluster configured to use the **ovs-networkpolicy** plugin, network isolation is controlled entirely by **NetworkPolicy** objects. By default, all pods in a project are accessible from other pods and network endpoints. To isolate one or more pods in a project, you can create **NetworkPolicy** objects in that project to indicate the allowed incoming connections. Project administrators can create and delete **NetworkPolicy** objects within their own project.

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。

Following are a few sample **NetworkPolicy** object definitions supporting different scenrios:

- **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、つまり、**同じ** namespace の Pod からのすべての接続、および **すべて** の namespace の Pod からのポート **80 443** での接続を受け入れます。

6.8.1. NetworkPolicy およびルーター

When using the **ovs-multitenant** plugin, traffic from the routers is automatically allowed into all namespaces. This is because the routers are usually in the **default** namespace, and all namespaces allow connections from pods in that namespace. With the **ovs-networkpolicy** plugin, this does not happen automatically. Therefore, if you have a policy that isolates a namespace by default, you need to take additional steps to allow routers to access it.

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。

Alternatively, you can create a policy allowing full access from the default namespace, as in the **ovs-multitenant** plugin:

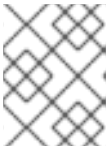
1. ラベルをデフォルト namespace に追加します。

重要

You only need to do this once for the entire cluster. The cluster administrator role is required to add labels to namesapces.

```
$ 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

Perform this step for each namespace you want to allow conntections into. Users with the Project Administrator role can create policies.

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-from-default-namespace
spec:
  podSelector:
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          name: default
```

6.8.2. 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. Create a custom project template and configure the master to use it, as described in Modifying the Template for New Projects.

2. 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
...
```

6.9. 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。

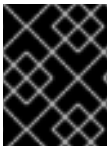
HSTS が有効にされると、HSTS はサイトから Strict Transport Security ヘッダーを HTTPS 応答に追加します。リダイレクトするルートで **insecureEdgeTerminationPolicy** 値を使用し、HTTP を HTTPS に送信するようにします。ただし、HSTS が有効にされている場合は、要求の送信前にクライアントがすべての要求を HTTP URL から HTTPS に変更するためにリダイレクトの必要がなくなります。これはクライアントでサポートされる必要はなく、**max-age=0** を設定することで無効にできます。

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。

6.10. 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ースが他の 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 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7章 サービスアカウントの設定

7.1. 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。

- 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。

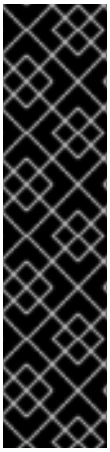
7.2. ユーザー名およびグループ

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。以下は例になります。

```
$ 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以下の 2 つのグループのメンバーでもあります。

system:serviceaccount
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:serviceaccount:<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:serviceaccount -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:serviceaccount:managers -n top-secret
```

7.3. 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に API トークンとレジストリーの認証情報が常にあることを確認します。

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。

7.4. 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。

7.5. 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. A **builder** service account in each project is required by build pods, and is given the **system:image-builder** role, which allows pushing images to any image stream in the project using the internal container registry.
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。

All service accounts in a project are given the **system:image-puller** role, which allows pulling images from any image stream in the project using the internal container registry.

7.6. 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
```

7.7. 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8章 ロールベースアクセス制御 (RBAC) の管理

8.1. 概要

You can use the CLI to view RBAC resources and the administrator CLI to manage the roles and bindings.

8.2. VIEWING ROLES AND BINDINGS

Roles can be used to grant various levels of access both cluster-wide as well as at the project-scope. Users and groups can be associated with, or **bound** to, multiple roles at the same time. You can view details about the roles and their bindings using the **oc describe** command.

Users with the **cluster-admin**default cluster role bound cluster-wide can perform any action on any resource. Users with the **admin** default cluster role bound locally can manage roles and bindings in that project.

注記

Review a full list of verbs in the Evaluating Authorization section.

8.2.1. Viewing Cluster Roles

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
```

Viewing Cluster Roles

```
$ 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]

...
```

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
```

Viewing Cluster Role Bindings

```
$ 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

...
```

8.2.2. Viewing Local Roles and Bindings

All of the default cluster roles can be bound locally to users or groups.

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
```

By default, the current project is used when viewing local role bindings. Alternatively, a project can be specified with the **-n** flag. This is useful for viewing the local role bindings of another project, if the user already has the **admin**default cluster role in it.

Viewing Local Role Bindings

```
$ 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
```

8.3. MANAGING ROLE BINDINGS

Adding, or **binding**, a role to users or groups gives the user or group the relevant access granted by the role. You can add and remove roles to and from users and groups using **oc adm policy** commands.

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8.1 Local Role Binding Operations

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。

コマンド	説明
$ 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。
$ 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。

You can also manage cluster role bindings using the following operations. The **-n** flag is not used for these operations because cluster role bindings uses non-namespaced resources.

表8.2 Cluster Role Binding Operations

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。

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
```

```
  User alice 1


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 The **alice** user has been added to the **admins RoleBinding**.

8.4. CREATING A LOCAL ROLE

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 ...
```

The following excerpt from the help of this command describes its usage:

```
Create a role with single rule.

Usage:
  oc create role NAME --verb=verb --resource=resource.group/subresource [--resource-
name=resourcename] [--dry-run] [options]
```

```
Examples:
  # Create a Role named "pod-reader" that allows user to perform "get", "watch" and "list" on pods
  oc create role pod-reader --verb=get --verb=list --verb=watch --resource=pods

  # Create a Role named "pod-reader" with ResourceName specified
  oc create role pod-reader --verb=get,list,watch --resource=pods --resource-name=readablepod --
resource-name=anotherpod

  # Create a Role named "foo" with API Group specified
  oc create role foo --verb=get,list,watch --resource=rs.extensions

  # Create a Role named "foo" with SubResource specified
  oc create role foo --verb=get,list,watch --resource=pods,pods/status

Options:
      --dry-run=false: If true, only print the object that would be sent, without sending it.
      --resource=[]: resource that the rule applies to
      --resource-name=[]: resource in the white list that the rule applies to, repeat this flag for multiple
items
      --verb=[]: verb that applies to the resources contained in the rule

...
```

For example, to create a role that allows a user to view pods, run:

```
$ oc create role podview --verb=get --resource=pod -n bob-project
```

Optionally, annotate it with a description.

To bind the new role to a user, run:

```
$ oc adm policy add-role-to-user podview user2 --role-namespace=bob-project -n bob-project
```

8.5. CLUSTER AND LOCAL ROLE BINDINGS

A cluster role binding is a binding that exists at the cluster level. A role binding exists at the project level. The cluster role **view** must be bound to a user using a local role binding for that user to view the project. Local roles should only created if a cluster role does not provide the set of permissions needed for a particular situation.

Some cluster role names are initially confusing. The cluster role **clusteradmin** can be bound to a user using a local role binding, making it appear that this user has the privileges of a cluster administrator. This is not the case. The **clusteradmin** cluster role bound to a certain project is more like a super administrator for that project, granting the permissions of the cluster role **admin**, plus a few additional permissions like the ability to edit rate limits. This can appear especially confusing via the web console UI, which does not list cluster role bindings (which are bound to true cluster administrators). However, it does list local role bindings (which could be used to locally bind **clusteradmin**).

第9章 イメージポリシー

9.1. 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。

Allowed Registries for import is an image policy configuration that allows to restrict image origins to particular set of external registries. This set of rules is applied to any image being imported or tagged into any image stream. Therefore any image referencing registry not matched by the rule set will be rejected.

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。
- **Image annotation restrictions**: limits or requires the annotations on an image in the integrated container registry

9.2. インポート用に許可されるレジストリーの設定

You can configure registries allowed for import in **master-config.yaml** under **imagePolicyConfig:allowedRegistriesForImport** section as demonstrated in the following example. If the setting is not present, all images are allowed.

例9.1 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.access.redhat.com 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構成されています。

- **domainName**: ホスト名であり、オプションでその最後は **:<port>** サフィックスになり、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。ワイルドカードは区切り文字の前または後の部分に適用されます。

- **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、サフィックスが **: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、サフィックスが **: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.access.redhat.com:443"
* status.tags[latest].items[0].dockerImageReference: Forbidden: registry "local.registry.corp" not
allowed by whitelist: "local.registry.corp:5000", "*.mydomain.com:80",
"registry.access.redhat.com:443"
```

9.3. IMAGEPOLICY 受付プラグインの設定

To enable this feature, configure the plug-in in **master-config.yaml**:

例9.2 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
    - targetResource: (8)
        group: batch
        resource: jobs
      localNames: true (9)
```

(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) Array of rules to evaluate against incoming resources. If you only have reject==true rules, the default is **allow all**. If you have any accept rule, the default is **deny all**.

(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** will allow single segment names (for example, **ruby:2.4**) to be interpreted as namespace-local image stream tags, but only if the resource or target image stream has **local name resolution** enabled.

注記

If you normally rely on infrastructure images being pulled using a default registry prefix (such as **docker.io** or **registry.access.redhat.com**), those images will not match to any **matchRegistries** value since they will have no registry prefix. To ensure infrastructure images have a registry prefix that can match your image policy, set the imageConfig.format value in your **master-config.yaml** file.

9.4. 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。以下は例になります。

```
$ oc annotate
images/sha256:09ce3d8b5b63595ffca6636c7daefb1a615a7c0e3f8ea68e5db044a9340d6ba8
images.openshift.io/deny-execution=true
```

7. この Pod を再作成します。Pod は拒否されます。

```
apiVersion: v1
kind: Pod
metadata:
  generateName: test-pod
spec:
  containers:
  - image: <internal registry IP>:5000/<namespace>/image-policy-check:latest
    name: first
```

第10章 イメージの署名

10.1. 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。

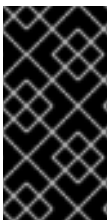
注記

Initial support for storing image signatures was added in OpenShift Container Platform 3.3. Initial support for verifying image signatures was added in OpenShift Container Platform 3.6.

10.2. 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。

The **atomic** command line interface (CLI), version 1.12.5 or greater, provides commands for signing container images, which can be pushed to an OpenShift Container Registry. The **atomic** CLI is available on Red Hat-based distributions: RHEL, Centos, and Fedora. The **atomic** CLI is pre-installed on RHEL Atomic Host systems. For information on installing the **atomic** package on a RHEL host, see Enabling Image Signature Support.



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>
```

Signatures are stored in OpenShift Container Platform when the **atomic** transport type argument is specified. See Signature Transports for more information.

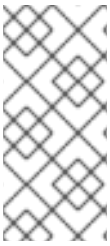
atomic CLI を使用してイメージをセットアップし、実行する方法についての詳細は、「RHEL Atomic Host Managing Containers: Signing Container Images」ドキュメントか、または **atomic push --help** 出力で引数の詳細を参照してください。

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。

10.3. 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>
```

Using the **--save** flag on already verified image together with invalid GPG key or invalid expected identity causes the saved verification status to be removed, and the image will become unverified.

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]
```

The **<pull_spec>** can be found by describing the image stream. The **<image>** may be found by describing the image stream tag. See the following example command output.

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
```

10.4. 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

The OpenShift Container Registry provides an **extensions** endpoint that allows you to write and read image signatures. The image signatures are stored in the OpenShift Container Platform key-value store via the Docker Registry API.

注記

This endpoint is experimental and not supported by the upstream Docker Registry project. See the upstream API documentation for general information about the Docker Registry API.

10.4.1. 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、**<digest>@<name>** 形式の一意の値である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。署名の名前には 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。

10.4.2. 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ことを仮定した場合、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、**<digest>@<name>** 形式の一意の値である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。署名の名前には 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。

10.4.3. 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 イメージについ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.access.redhat.com:
    sigstore: https://access.redhat.com/webassets/docker/content/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。

For more details about the registries configuration directory, see Registries Configuration Directory.

第11章 スコープ付きトークン

11.1. 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。

11.2. 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。

11.3. 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** - Allows full read/write access to the API with all of the user's permissions.
- **user:info** - Allows read-only access to information about the user: name, groups, and so on.
- **user:check-access** - Allows access to **self-localsubjectaccessreviews** and **self-subjectaccessreviews**. These are the variables where you pass an empty user and groups in your request object.
- **user:list-projects** - Allows read-only access to list the projects the user has access to.

11.4. 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12章 イメージのモニタリング

12.1. 概要

You can monitor images in your instance using the CLI.

12.2. イメージ統計の表示

OpenShift Container Platform can display several usage statistics about all the images it manages. In other words, all the images pushed to the internal registry either directly or through a build.

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。

- image ID
- project, name, and tag of the accompanying **ImageStreamTag**
- potential parents of the image, using their ID
- information about where the image is being used
- flag informing whether the image contains proper Docker metadata information
- size of the image

12.3. IMAGESTREAMS 統計の表示

OpenShift Container Platform can display several usage statistics about all the **ImageStreams**.

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。

- project and name of the **ImageStream**
- size of the entire **ImageStream** stored in the internal Red Hat Container Registry
- number of images this particular **ImageStream** is pointing to
- number of layers **ImageStream** consists of

12.4. イメージのプルーニング

The information returned from the above commands is helpful when performing image pruning.

第13章 SCC (SECURITY CONTEXT CONSTRAINTS) の管理

13.1. 概要

Security context constraints allow administrators to control permissions for pods. To learn more about this API type, see the security context constraints (SCCs) architecture documentation. You can manage SCCs in your instance as normal API objects using the CLI.

注記

You must have **cluster-admin** privileges to manage SCCs.

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。

13.2. 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]
```

13.3. SCC (SECURITY CONTEXT CONSTRAINTS) オブジェクトの検査

To examine a particular SCC, use **oc get**, **oc describe**, **oc export**, or **oc edit**. For example, to examine the **restricted** SCC:

```
$ oc describe scc restricted
Name:     restricted
Priority:    <none>
Access:
  Users:    <none>
  Groups:    system:authenticated
```

```
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>
```

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。

13.4. 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。

1. 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
```

2. 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 gitRepo glusterfs iscsi nfs
persistentVolumeClaim photonPersistentDisk quobyte rbd secret vsphere]
```

13.5. 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>
```

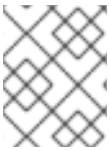
注記

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。

13.6. 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。

13.6.1. 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 When a container or pod does not request a user ID under which it should be run, the effective UID depends on the SCC that emits this pod. Because restricted SCC is granted to all authenticated users by default, it will be available to all users and service accounts and used in most cases. The restricted SCC uses **MustRunAsRange** strategy for constraining and defaulting the possible values of the **securityContext.runAsUser** field. The admission plug-in will look for the **openshift.io/sa.scc.uid-range** annotation on the current project to populate range fields, as it does not provide this range. In the end, a container will have **runAsUser** equal to the first value of the range that is hard to predict because every project has different ranges. See Understanding Pre-allocated Values and Security Context Constraints for more information.

明示的な runAsUser 設定がない場合

```
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。

This works with SELinux, fsGroup, and Supplemental Groups. See Volume Security for more information.

13.7. 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。

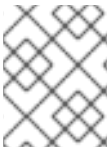
1. リセットする SCC を削除し、マスターを再起動してその再作成を実行します。
2. **oc adm policy reconcile-sccs** コマンドを使用します。

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。

13.8. 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。

13.8.1. 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。

13.8.2. 特権付き SCC のサービスアカウントアクセスの付与

First, create a service account. For example, to create service account **mysvcacct** in project **myproject**:

```
$ oc create serviceaccount mysvcacct -n myproject
```

次に、サービスアカウントを**特権付き** 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。

13.8.3. 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 で実行することができます。

13.8.4. 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。

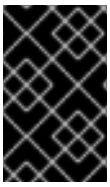
```
$ oc adm policy add-scc-to-user anyuid system:serviceaccount:myproject:mysvcacct
```

13.8.5. レジストリーでの --mount-host の使用

It is recommended that persistent storage using **PersistentVolume** and **PersistentVolumeClaim** objects be used for registry deployments. If you are testing and would like to instead use the **oc adm registry** command with the **--mount-host** option, you must first create a new service account for the registry and add it to the **privileged** SCC. See the Administrator Guide for full instructions.

13.8.6. 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。

13.8.7. クラスターのデフォルト動作の変更

To modify your cluster so that it does not pre-allocate UIDs, allows containers to run as any user, and prevents privileged containers:

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。

1. Edit the **restricted** SCC:

```
$ oc edit scc restricted
```

2. Change **runAsUser.Type** to **RunAsAny**.

3. Ensure **allowPrivilegedContainer** is set to false.
4. Save the changes.

To modify your cluster so that it does not pre-allocate UIDs and does not allow containers to run as root:

1. Edit the **restricted** SCC:

```
$ oc edit scc restricted
```

2. Change **runAsUser.Type** to **MustRunAsNonRoot**.
3. Save the changes.

13.8.8. hostPath ボリュームプラグインの使用

To relax the security in your cluster so that pods are allowed to use the **hostPath** volume plug-in without granting everyone access to the **privileged** SCC:

1. Edit the **restricted** SCC:

```
$ oc edit scc restricted
```

2. Add **allowHostDirVolumePlugin: true**.
3. Save the changes.

13.8.9. 受付を使用した特定 SCC の初回使用

You may control the sort ordering of SCCs in admission by setting the **Priority** field of the SCCs. See the SCC Prioritization section for more information on sorting.

13.8.10. SCC のユーザー、グループまたはプロジェクトへの追加

Before adding an SCC to a user or group, you can first use the **scc-review** option to check if the user or group can create a pod. See the Authorization topic for more information.

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、その使用をを強く推奨します。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14章 スケジューリング

14.1. 概要

14.1.1. 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。

14.1.2. 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。

14.1.3. 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。

14.1.4. 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。

14.2. デフォルトスケジューリング

14.2.1. 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。

14.2.2. 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。

14.2.3. 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。

14.2.3.1. 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ために管理者によって使用されます。

14.2.3.2. 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 Post 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。

14.2.4. 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"
        },
        {
            "name": "CheckNodeMemoryPressure"
        },
        {
            "name": "CheckNodeDiskPressure"
        },
        {
            "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
        }
    ]
}
```

14.2.4.1. スケジューラーポリシーの変更

The scheduler policy is defined in a file on the master, named **/etc/origin/master/scheduler.json** by default, unless overridden by the **kubernetesMasterConfig.schedulerConfigFile** field in the master configuration file.

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。

```
# systemctl restart atomic-openshift-master-api atomic-openshift-master-controllers
```

14.2.5. 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。

14.2.5.1. 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。

14.2.5.1.1. 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 がノードテイントを許容できるかどうかを確認します。

```
{"name" : "PodToleratesNodeTaints"}
```

CheckNodeMemoryPressure checks if a pod can be scheduled on a node with a memory pressure condition.

```
{"name" : "CheckNodeMemoryPressure"}
```

14.2.5.1.2. 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。

CheckNodeDiskPressure checks if a pod can be scheduled on a node with a disk pressure condition.

```
{"name" : "CheckNodeDiskPressure"}
```

CheckVolumeBinding は、バインドされている PVC とバインドされていない PVC の両方の場合に Pod が要求するボリュームに基づいて適しているかどうかを評価します* バインドされている PVC については、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によって満たされていることを確認します。* バインドされていない PVC については、述語は PVC 要件を満たす PV を検索し、PV のノードアフィニティーが指定ノードによって満たされていることを確認します。

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 Pod をノードでスケジュールできるかどうかを確認し、 **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)** 状態を報告します。

```
{"name" : "CheckNodeCondition"}
```

PodToleratesNodeNoExecuteTaints は、Pod 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を問わない)。

```
{"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"}
```

14.2.5.2. 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"}
```

汎用的なクリティカル述語
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"}
```

14.2.5.3. 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
      "weight" : "1" 2
      "argument":{
        "serviceAffinity":{
          "labels":[
            "<label>" 3
          ]
        }
      }
    }
  ],
```

1 述語の名前を指定します。

2 Specify a weight from 1 (bad fit) to 10 (best fit).

3 Specify a label for matching. For example:

```
      "name":"ZoneAffinity",
      "weight" : "1"
      "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 checks whether a particular node has a certain label defined or not, regardless of its value. Matching by label can be useful, for example, where nodes have their physical location or status defined by labels.

```
"predicates":[
    {
      "name":"<name>", 1
      "weight" : "1" 2
      "argument":{
        "labelsPresence":{
          "labels":[
            "<label>" 3
           presence: true/false
          ]
        }
      }
    }
],
```

1 述語の名前を指定します。

2 Specify a weight from 1 (bad fit) to 10 (best fit).

3 Specify a label for matching.

Specify whether the labels are required.

- **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。
- For **presence:true**, if all of the requested labels are present in the node labels, the pod can be scheduled. If all of the lables are not present, the pod is not scheduled.

例:

```
    "name":"RackPreferred",
    "weight" : "1"
    "argument":{
      "labelsPresence":{
        "labels":[
          "rack"
      "labelsPresence:"{
        "labels:"[
        - "region"
        presence: true
```

14.2.6. 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。

14.2.6.1. 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。

14.2.6.1.1. 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 Pod が少ないノードを優先し、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}
```

14.2.6.1.2. 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}
```

14.2.6.2. 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":{
            "labels":[
              "<label>" (3)
            ]
          }
        }
      }
    ]
```

(1) 優先度の名前を指定します。

(2) Specify a weight from 1 (bad fit) to 10 (best fit).

(3) Specify a label for matching.

例:

```
        "name":"RackSpread",
        "weight" : "1"
        "argument":{
          "serviceAffinity":{
            "labels":[
              "rack"
```

LabelPreference prefers nodes that have a particular label defined, regardless of its value.

```
"predicates":[
    {
      "name":"<name>", (1)
      "weight" : "1" (2)
      "argument":{
        "labelsPresence":{
          "labels":[
            "<label>" (3)
           presence: true/false
          ]
        }
      }
    }
  ],
```

(1) 優先度の名前を指定します。

(2) Specify a weight from 1 (bad fit) to 10 (best fit).

(3) Specify a label for matching.

Specify whether the labels are required.

- **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。
- For **presence:true**, if all of the requested labels are present in the node labels, the pod can be scheduled. If all of the lables are not present, the pod is not scheduled.

例:

```
"name":"RackPreferred",
"weight" : "1"
"argument":{
  "labelsPresence":{
    "labels":[
      "rack"
```

14.2.7. 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。

14.2.7.1. 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。さらに、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。

14.2.7.2. 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。

14.2.7.3. 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。

14.2.8. 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
```

14.3. カスタムスケジューリング

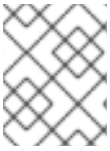
14.3.1. 概要

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。

14.3.2. Deploying the Scheduler

The steps below are the general process for deploying a scheduler into your cluster.



注記

Information on how to create/deploy a scheduler is outside the scope of this document. For an example, see **plugin/pkg/scheduler** in the Kubernetes source directory .

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
  labels:
    name: multischeduler-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 scheduler.yaml
```

3. Run the following command to check that the pod was created with the custom scheduler:

```
$ oc get pod custom-scheduler -o yaml
```

4. Run the following command to check the status of the pod:

```
$ oc get pod
```

   The pod should not be running.

```
NAME               READY     STATUS    RESTARTS   AGE
custom-scheduler   0/1       Pending   0          2m
```

5. Deploy the custom scheduler.

6. Run the following command to check the status of the pod:

```
$ oc get pod
```

The pod should be running.

```
NAME              READY     STATUS    RESTARTS   AGE
custom-scheduler  1/1       Running   0          4m
```

7. Run the following command to check that the scheduler was used:

```
$ oc describe pod custom-scheduler
```

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      my-scheduler     Normal         Scheduled  Successfully assigned
custom-scheduler to <$node1>
[...]
```

14.4. POD 配置の制御

14.4.1. 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。

The Pod Node Constraints admission controller ensures that pods are deployed onto only specified node hosts using labels] and prevents users without a specific role from using the **nodeSelector** field to schedule pods.

14.4.2. 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. Modify the master configuration file (**/etc/origin/master/master-config.yaml**) in two places:

   a. Add **PodNodeConstraints** to the **admissionConfig** section:

```
...
admissionConfig:
  pluginConfig:
    PodNodeConstraints:
```

```
      configuration:
        apiversion: v1
        kind: PodNodeConstraintsConfig
...
```

b. Then, add the same to the **kubernetesMasterConfig** section:

```
...
kubernetesMasterConfig:
  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。

```
# systemctl restart atomic-openshift-master
```

14.4.3. 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。

For example, an OpenShift Container Platform cluster might consist of five data centers spread across two regions. In the U.S., "us-east", "us-central", and "us-west"; and in the Asia-Pacific region (APAC), "apac-east" and "apac-west". Each node in each geographical region is labeled accordingly. For example, **region: us-east**.

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. Modify the master configuration file (**/etc/origin/master/master-config.yaml**) in two places:

   a. Add **nodeSelectorLabelBlacklist** to the **admissionConfig** section with the labels that are assigned to the node hosts you want to deny pod placement:

```
...
admissionConfig:
  pluginConfig:
    PodNodeConstraints:
      configuration:
        apiversion: v1
        kind: PodNodeConstraintsConfig
        nodeSelectorLabelBlacklist:
          - kubernetes.io/hostname
          - <label>
...
```

   b. Then, add the same to the **kubernetesMasterConfig** section to restrict direct pod creation:

```
...
kubernetesMasterConfig:
  admissionConfig:
    pluginConfig:
      PodNodeConstraints:
        configuration:
          apiVersion: v1
          kind: PodNodeConstraintsConfig
          nodeSelectorLabelBlacklist:
            - kubernetes.io/hostname
            - <label_1>
...
```

3. 変更を有効にするために OpenShift Container Platform を再起動します。

```
# systemctl restart atomic-openshift-master
```

14.4.4. プロジェクト対する Pod 配置の制御

The **Pod Node Selector** admission controller allows you to force pods onto nodes associated with a specific project and prevent pods from being scheduled in those nodes.

The **Pod Node Selector** admission controller determines where a pod can be placed using labels on projects and node selectors specified in pods. A new pod will be placed on a node associated with a project only if the node selectors in the pod match the labels in the project.

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。

The **Pod Node Selector** admission controller also allows you to create a list of labels that are permitted in a specific project. This list acts as a **whitelist** that lets developers know what labels are acceptable to use in a project and gives administrators greater control over labeling in a cluster.

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。

   - Add the following to the master configuration file (**/etc/origin/master/master-config.yaml**):

     ```
     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) (3) すべてのノードのデフォルトラベルを作成します。

     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

If a project does not have a node selectors specified, the pods associated with that project will be merged using the default node selector (**clusterDefaultNodeSelector**).

2. 変更を有効にするために OpenShift Container Platform を再起動します。

```
# systemctl restart atomic-openshift-master
```

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。

```
{
    "kind": "Namespace",
    "apiVersion": "v1",
    "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。

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です。

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** はにずれのプロジェクト仕様にもありません。

```
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。

14.5. 詳細スケジューリング

14.5.1. 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。

通常、詳細スケジューリングは必要になりません。OpenShift Container Platform が Pod を合理的な方法で自動的に配置するためです。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必要がある場合があります。

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。

14.5.2. 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する制御を行いません。
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する制御を行いません。
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が、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。

14.6. 詳細スケジューリングおよびノードのアフィニティー

14.6.1. 概要

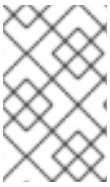
Node affinity is a set of rules used by the scheduler to determine where a pod can be placed. The rules are defined using custom labels on nodes and label selectors specified in pods. Node affinity allows a

pod to specify an affinity (or anti-affinity) towards a group of **nodes** it can be placed on. The node does not have control over the placement.

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。

required (必須) ルールは、Pod をノードにスケジュールする前に**満たされている必要があります**。一方、preferred (優先) ルールは、ルールが満たされる場合にスケジューラーが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。

14.6.2. 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。

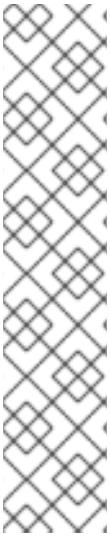
2 preferred (優先) ルールを定義します。

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 ノードセレクターを使用している場合、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。

14.6.2.1. ノードアフィニティーの required (必須) ルールの設定

Pod がノードにスケジュールされる前に、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
```

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. 満たす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で必要になるようにします。

```
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
```

14.6.2.2. 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
```

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. 満たす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
```

14.6.3. 例

以下の例は、ノードのアフィニティーを示しています。

14.6.3.1. 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
```

14.6.3.2. 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
<---snip--->
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).
```

14.7. 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

14.7.1. 概要

Pod affinity and **pod anti-affinity** allow you to specify rules about how pods should be placed relative to other pods. The rules are defined using custom labels on nodes and label selectors specified in pods. Pod affinity/anti-affinity allows a **pod** to specify an affinity (or anti-affinity) towards a group of **pods** it can be placed with. The node does not have control over the placement.

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 affinity/anti-affinity allows you to constrain which nodes your pod is eligible to be scheduled on based on the labels on other pods. A label is a key/value pair.

- 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。

required (必須) ルールは、Pod をノードにスケジュールする前に **満たされている必要があります**。一方、preferred (優先) ルールは、ルールが満たされる場合にスケジューラーが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。

14.7.2. 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 6
      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 Specifies a weight for a preferred rule. The node that with highest weight is preferred.

4 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と値 (ラベル) です。

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。

14.7.2.1. 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
```

14.7.2.2. 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
```

14.7.3. 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。

14.7.3.1. 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。

14.7.3.2. 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: 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、**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>
```

14.7.3.3. 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 **pod-s1** と同じノードにスケジュールできません。

14.8. 詳細スケジューリングおよびノードセレクター

14.8.1. 概要

A **node selector** specifies a map of key-value pairs. The rules are defined using custom labels on nodes and selectors specified in pods.

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。

14.8.2. 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. Modify the master configuration file (**/etc/origin/master/master-config.yaml**) in two places:

   a. Add **nodeSelectorLabelBlacklist** to the **admissionConfig** section with the labels that are assigned to the node hosts you want to deny pod placement:

```
...
admissionConfig:
  pluginConfig:
    PodNodeConstraints:
      configuration:
        apiversion: v1
        kind: PodNodeConstraintsConfig
        nodeSelectorLabelBlacklist:
          - kubernetes.io/hostname
          - <label>
...
```

   b. Then, add the same to the **kubernetesMasterConfig** section to restrict direct pod creation:

```
...
kubernetesMasterConfig:
  admissionConfig:
    pluginConfig:
      PodNodeConstraints:
        configuration:
          apiVersion: v1
          kind: PodNodeConstraintsConfig
          nodeSelectorLabelBlacklist:
```

```
            - kubernetes.io/hostname
            - <label_1>
...
```

3. 変更を有効にするために OpenShift Container Platform を再起動します。

```
# systemctl restart atomic-openshift-master
```

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。

14.9. 詳細スケジューリングおよび容認

14.9.1. 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。

14.9.2. 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.で構成されています。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14.1 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。

<table>
<tr><th>パラメーター</th><th>説明</th></tr>
<tr><td>value</td><td>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。</td></tr>
<tr><td>effect</td><td>effect は以下のいずれかにすることができます。<table>
<tr><td>NoSchedule</td><td>テイントに一致しない新規 Pod はノードにスケジュールされません。
ノードの既存 Pod はそのままになります。</td></tr>
<tr><td>PreferNoSchedule</td><td>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。</td></tr>
<tr><td>NoExecute</td><td>テイントに一致しない新規 Pod はノードにスケジュールできません。
一致する容認を持たないノードの既存 Pod は削除されます。</td></tr>
</table></td></tr>
<tr><td>operator</td><td><table>
<tr><td>Equal</td><td>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。</td></tr>
<tr><td>Exists</td><td>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。</td></tr>
</table></td></tr>
</table>

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。

14.9.2.1. 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。

例:

- 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。

14.9.3. 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>
```

例:

```
$ oc adm taint nodes node1 key1=value1:NoSchedule
```

The example places a taint on **node1** that has key **key1**, value **value1**, and taint effect **NoSchedule**.

14.9.4. 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。

14.9.4.1. 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。

例:

```
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。

14.9.4.1.1. 容認の秒数のデフォルト値の設定

This plug-in sets the default forgiveness toleration for pods, to tolerate the **node.alpha.kubernetes.io/notReady:NoExecute** and **node.alpha.kubernetes.io/notReady:NoExecute** taints for five minutes.

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。

```
# systemctl restart atomic-openshift-master-api atomic-openshift-master-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>
```

例:

```
$ 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
```

例:

```
$ oc describe pod hello-pod |grep -i toleration
Tolerations:    node.alpha.kubernetes.io/notReady=:Exists:NoExecute for 300s
```

14.9.5. Preventing Pod Eviction for Node Problems

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。

- If a node enters a not ready state, the **node.alpha.kubernetes.io/notReady:NoExecute** taint is added and pods cannot be scheduled on the node. Existing pods remain for the toleration seconds period.
- If a node enters a not reachable state, the **node.alpha.kubernetes.io/unreachable:NoExecute** taint is added and pods cannot be scheduled on the node. Existing pods remain for the toleration seconds period.

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.alpha.kubernetes.io/notReady:NoExecute
```

3. 変更を有効にするために、OpenShift を再起動します。

```
# systemctl restart atomic-openshift-master-api atomic-openshift-master-controllers
```

4. 容認を Pod に追加します。

```
tolerations:
- key: "node.alpha.kubernetes.io/unreachable"
  operator: "Exists"
  effect: "NoExecute"
  tolerationSeconds: 6000
```

または、以下を実行します。

```
tolerations:
- key: "node.alpha.kubernetes.io/notReady"
  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。

14.9.6. Daemonset および容認

DaemonSet pods are created with **NoExecute** tolerations for **node.alpha.kubernetes.io/unreachable** and **node.alpha.kubernetes.io/notReady** with no **tolerationSeconds** to ensure that DaemonSet pods are never evicted due to these problems, even when the Default Toleration Seconds feature is disabled.

14.9.7. 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

14.9.7.1. 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。
例:

```
$ oc adm taint nodes node1 dedicated=groupName:NoSchedule
```

2. Add a corresponding toleration to the pods by writing a custom admission controller.
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。

14.9.7.2. 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。
例:

```
$ oc adm taint nodes node1 dedicated=groupName:NoSchedule
```

2. Add a corresponding toleration to the pods by writing a custom admission controller.
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。

14.9.7.3. 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. Adding a corresponding toleration to pods that use the special hardware using an admission controller.

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15章 クォータの設定

15.1. 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

See the Developer Guide for more on compute resources.

15.2. 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15.1 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、交換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、交換可能なものとして使用できます。
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、交換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、交換可能なものとして使用できます。
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。

表15.2 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 Persistent Volume Claim (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 Persistent Volume Claim (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 Persistent Volume Claim (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15.3 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。
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 Persistent Volume Claim (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。

15.3. 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**

15.4. 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。

15.5. REQUESTS VS LIMITS

When allocating compute resources, each container may specify a request and a limit value each for CPU and memory. Quotas can restrict any of these values.

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。

15.6. 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)
    limits.cpu: "2" (4)
    limits.memory: 2Gi (5)
```

(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、CPU 制限の合計は 2 コア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、メモリー制限の合計は 2 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)
  scopes:
  - NotTerminating (4)
```

(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 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
  scopes:
  - Terminating 4
```

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 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 Persistent Volume Claim (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 Persistent Volume Claim (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 Persistent Volume Claim (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 Persistent Volume Claim (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 Persistent Volume Claim (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 Persistent Volume Claim (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 Persistent Volume Claim (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。

15.7. クォータの作成

To create a quota, first define the quota to your specifications in a file, for example as seen in Sample Resource Quota Definitions. Then, create using that file to apply it to a project:

```
$ oc create -f <resource_quota_definition> [-n <project_name>]
```

例:

```
$ oc create -f resource-quota.json -n demoproject
```

15.8. 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、以下を実行します。

```
$ oc get quota -n demoproject
NAME                AGE
besteffort          11m
compute-resources   2m
core-object-counts  29m
```

2. 次に、関連す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
```

15.9. 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。

```
# systemctl restart atomic-openshift-master-api atomic-openshift-master-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。

15.10. 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

If a quota has been defined for your project, see Deployment Resources for considerations on any deployment configurations.

15.11. 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。

第16章 複数プロジェクトのクォータ設定

16.1. 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、クォータを複数プロジェクト間で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。

16.2. 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
```

16.3. 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
```

16.4. 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17章 制限範囲の設定

17.1. 概要

A limit range, defined by a **LimitRange** object, enumerates compute resource constraints in a project at the pod, container, image, image stream, and persistent volume claim level, and specifies the amount of resources that a pod, container, image, image stream, or persistent volume claim can consume.

All resource create and modification requests are evaluated against each **LimitRange** object in the project. If the resource violates any of the enumerated constraints, then the resource is rejected. If the resource does not set an explicit value, and if the constraint supports a default value, then the default value is applied to the resource.

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) The minimum amount of CPU that a pod can request on a node across all containers.

(5) The minimum amount of memory that a pod can request on a node across all containers.

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 The minimum amount of CPU that a single container in a pod can request.

9 The minimum amount of memory that a single container in a pod can request.

10 The default amount of CPU that a container will be limited to use if not specified.

11 The default amount of memory that a container will be limited to use if not specified.

12 The default amount of CPU that a container will request to use if not specified.

13 The default amount of memory that a container will request to use if not specified.

14 The maximum amount of CPU burst that a container can make as a ratio of its limit over request.

For more information on how CPU and memory are measured, see Compute Resources.

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)
```

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です。

2 The maximum number of unique image tags per image stream's spec.

3 The maximum number of unique image references per image stream's status.

Both core and OpenShift Container Platform resources can be specified in just one limit range object. They are separated here into two examples for clarity.

17.1.1. 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17.1 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 If the configuration defines a **min** CPU, then the request value must be greater than the CPU value. A limit value does not need to be specified.
Max	**container.resources.limits[resource]** (必須): **Max[resource]** 以下 If the configuration defines a **max** CPU, then you do not need to define a request value, but a limit value does need to be set that satisfies the maximum CPU constraint.
MaxLimitRequestRatio	**MaxLimitRequestRatio[resource]** less than or equal to (**container.resources.limits[resource]** / **container.resources.requests[resource]**) If a configuration defines a **maxLimitRequestRatio** value, then any new containers must have both a request and limit value. Additionally, OpenShift Container Platform calculates a limit to request ratio by dividing the limit by the request. For example, if a container has **cpu: 500** in the **limit** value, and **cpu: 100** in the **request** value, then its limit to request ratio for**cpu** is **5**. This ratio must be less than or equal to the **maxLimitRequestRatio**.

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。

17.1.2. 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17.2 Pod

制約	実施される動作
Min	**Min[resource]** less than or equal to **container.resources.requests[resource]** (required) less than or equal to **container.resources.limits[resource]** (optional)
Max	**container.resources.limits[resource]** (必須): **Max[resource]** 以下
MaxLimitRequestRatio	**MaxLimitRequestRatio[resource]** less than or equal to (**container.resources.limits[resource]** / **container.resources.requests[resource]**)

17.1.3. 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17.3 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

To prevent blobs exceeding the limit from being uploaded to the registry, the registry must be configured to enforce quota. An environment variable **REGISTRY_MIDDLEWARE_REPOSITORY_OPENSHIFT_ENFORCEQUOTA** must be set to **true** which is done by default for new deployments. To update older deployment configuration, refer to Enforcing quota in the Registry .

警告

The image size is not always available in the manifest of an uploaded image. This is especially the case for images built with Docker 1.10 or higher and pushed to a v2 registry. If such an image is pulled with an older Docker daemon, the image manifest will be converted by the registry to schema v1 lacking all the size information. No storage limit set on images will prevent it from being uploaded.

現在、この問題への対応が行われています。

17.1.4. 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17.4 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** returns unique image names found in status tags. The name equals image's digest.

17.1.4.1. イメージ参照の数

Resource **openshift.io/image-tags** represents unique image references. Possible references are an **ImageStreamTag**, an **ImageStreamImage** and a **DockerImage**. They may be created using commands **oc tag** and **oc import-image** or by using tag tracking. No distinction is made between internal and external references. However, each unique reference tagged in the image stream's specification is counted just once. It does not restrict pushes to an internal container registry in any way, but is useful for tag restriction.

Resource **openshift.io/images** represents unique image names recorded in image stream status. It allows for restriction of a number of images that can be pushed to the internal registry. Internal and external references are not distinguished.

17.1.5. 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17.5 Pod

制約	実施される動作
Min	Min[resource] ⇐ claim.spec.resources.requests[resource] (required)
Max	claim.spec.resources.requests[resource] (required) ⇐ 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) The minimum amount of storage that can be requested in a persistent volume claim

(3) The maximum amount of storage that can be requested in a persistent volume claim

17.2. 制限範囲の作成

To apply a limit range to a project, create a limit range object definition on your file system to your desired specifications, then run:

```
$ oc create -f <limit_range_file> -n <project>
```

17.3. VIEWING LIMITS

You can view any limit ranges defined in a project by navigating in the web console to the project's **Quota** page.

You can also use the CLI to view limit range details:

1. First, get the list of limit ranges defined in the project. For example, for a project called **demoproject**:

   -

```
$ oc get limits -n demoproject
NAME              AGE
resource-limits   6d
```

2. Then, describe the limit range you are interested in, for example the **resource-limits** limit range:

```
$ oc describe limits resource-limits -n demoproject
Name:                   resource-limits
Namespace:              demoproject
Type                    Resource              Min     Max     Default Request Default Limit   Max
Limit/Request Ratio
----                    --------              ---     ---     --------------- -------------   -----------------------
Pod                     cpu                   200m    2       -               -               -
Pod                     memory                6Mi     1Gi     -               -               -
Container               cpu                   100m    2       200m            300m            10
Container               memory                4Mi     1Gi     100Mi           200Mi           -
openshift.io/Image      storage               -       1Gi     -               -               -
openshift.io/ImageStream        openshift.io/image      -       12      -               -               -
openshift.io/ImageStream        openshift.io/image-tags -       10      -               -               -
```

17.4. DELETING LIMITS

Remove any active limit range to no longer enforce the limits of a project:

```
$ oc delete limits <limit_name>
```

第18章 PRUNING OBJECTS

18.1. 概要

Over time, API objects created in OpenShift Container Platform can accumulate in the etcd data store through normal user operations, such as when building and deploying applications.

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。

18.2. BASIC PRUNE OPERATIONS

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。

- The **<object_type>** to perform the action on, such as **builds**, **deployments**, or **images**.
- 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>**。

18.3. PRUNING DEPLOYMENTS

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18.1 Prune Deployments CLI Configuration Options

オプション	説明
--confirm	ドライランの実行ではなく、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**)。

オプション	説明
--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**)。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
```

18.4. PRUNING BUILDS

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18.2 Prune Builds CLI Configuration Options

オプション	説明
--confirm	ドライランの実行ではなく、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

Developers can enable automatic build pruning by modifying their build configuration.

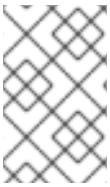
18.5. イメージのプルーニング

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>]
```

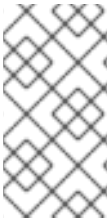
注記

Currently, to prune images you must first log in to the CLI as a user with an access token. The user must also have the cluster role**system:image-pruner** or greater (for example, **cluster-admin**).



注記

Pruning images removes data from the integrated registry unless **--prune-registry=false** is used. For this operation to work properly, ensure your registry is configured with **storage:delete:enabled** set to **true**.



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。したがって、プルーニング後はキャッシュをクリアする必要があります。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
```

If the integrated registry uses a redis cache, you need to clean the database manually.

If redeploying the registry after pruning is not an option, then you must permanently disable the cache.

表18.3 Prune Images CLI Configuration Options

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	Indicate that pruning should occur, instead of performing a dry-run. This requires a valid route to the integrated Docker registry. If this command is run outside of the cluster network, the route needs to be provided using **--registry-url**.
--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。

18.5.1. Image Prune Conditions

- **--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 とその後に レジストリーのハードプルーニング を実行することで、OpenShift Container Platform イメージ API オブジェクトの削除とイメージデータのレジストリーからの削除を分離することができます。これによりタイミングウィンドウが制限され、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
```

18.5.2. Using Secure or Insecure Connections

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。

18.5.3. Image Pruning Problems

Images Not Being Pruned
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、プルーニング候補のイメージについて満たすべき条件があることを確認します。

Especially ensure that images you want removed occur at higher positions in each tag history than your chosen tag revisions threshold. For example, consider an old and obsolete image named **sha:abz**. By running the following command in namespace **N**, where the image is tagged, you will see the image is tagged three times in a single image stream named **myapp**:

```
$ image_name="sha:abz"
$ oc get is -n N -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}}\n'\
  '{{end}}{{end}}{{end}}{{end}}'
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. Delete all the **istags** where the position is below the revision threshold, which means **myapp:v2.1** and **myapp:v2.1-may-2016**.

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。

Tags having a date or time of a particular image's build in their names should be avoided, unless the image needs to be preserved for undefined amount of time. Such tags tend to have just one image in its history, which effectively prevents them from ever being pruned. Learn more about **istag** naming.

Using a Secure Connection Against Insecure Registry
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. The recommened solution is to secure the registry. If that is not desired, you can force the client to use an insecure connection by appending **--force-insecure** to the command **(not recommended)**.

18.5.3.1. Using an Insecure Connection Against a Secured Registry

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。

Use the **--certificate-authority** option to provide the right certificate authority for the Docker registry server.

Using the Wrong Certificate Authority
The following error means that the certificate authority used to sign the certificate of the secured Docker registry is different than the authority used by the client.

```
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)**。

18.6. HARD PRUNING THE REGISTRY

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ない。

- **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 すべては孤立する。
- OpenShift Container Platform がクォータの制限によりイメージを拒否する。
- 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る。
- 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. **Log in**: Log in using the CLI as a user with an access token.
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。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. Set the following environment variable:

   ```
   $ oc 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。再デプロイが完了するのを待機してから次に進んでください。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。これにより変更が加えられることはありません。

   ```
   $ 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'
   ```

   ### Sample Output (Truncated)

   ```
   $ 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
```

Sample Output

```
$ 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 env -n default dc/docker-registry
REGISTRY_STORAGE_MAINTENANCE_READONLY-
```

18.7. 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。これらの機能は、近々発表予定の製品機能をリリースに先駆けてご提供することにより、お客様は機能性をテストし、開発プロセス中にフィードバックをお寄せいただくことができます。

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 jobs can perform pruning of successful jobs, but might not handle properly, the failed jobs. Therefore, cluster administrator should perform regular cleanup of jobs, manually. We also recommend to restrict the access to cron jobs to a small group of trusted users and set appropriate quota to prevent the cron job from creating too many jobs and pods.

第19章 EXTENDING THE KUBERNETES API WITH CUSTOM RESOURCES

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。

19.1. CREATING CUSTOM RESOURCE DEFINITIONS

To create a CRD, open a YAML file and enter the fields in the following example.

Example YAML file for a Custom Resource Definition

```
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. Specify a version name to be used in the URL. Each API Group can exist in multiple versions. For example: **v1alpha**, **vibeta**, **v1**.

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. Specify the plural name to be used in the URL. The **plural** field is the same as a resource in an API URL.
7. Specify a singular name to be used as an alias on the CLI and for display.
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。

After configuring the definition file, create the object:

```
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>/...
```

For example, using the example file, the following endpoint would be created:

```
/apis/stable.example.com/v1/namespaces/*/crontabs/...
```

This endpoint URL can then be used to create and manage custom objects. The kind of object is based on the **spec.kind** field of the Custom Resource Definition object you created.

19.2. CREATE CUSTOM OBJECTS

After the custom resource definition object has been created, you can create custom objects.

Custom objects can contain custom fields. These fields can contain arbitrary JSON.

In the following example, the **cronSpec** and **image** custom fields are set in a custom object of kind **CronTab**. The kind **CronTab** comes from the **spec.kind** field of the custom resource definition object you created above.

Example YAML file for a Custom Object

```
apiVersion: "stable.example.com/v1" 1
kind: CronTab 2
metadata:
  name: my-new-cron-object 3
spec: 4
  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. オブジェクトのタイプに固有の条件を指定します。

After configuring the object file, create the object:

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

19.3. MANAGE CUSTOM OBJECTS

You can then manage your custom resources.

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>
```

例:

```
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。以下は例になります。

```
oc get crontabs
oc get crontab
oc get ct
```

You can also view the raw JSON data:

```
oc get <kind> -o yaml
```

You should see that it contains the custom <1> **cronSpec** and <2> **image** fields from the YAML you used to create it:

```
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
```

19.4. FINALIZERS

Custom objects support **finalizers**, which allow controllers to implement conditions that must be completed before the object can be deleted.

You can add a finalizer to a custom object like this:

```
apiVersion: "stable.example.com/v1"
kind: CronTab
metadata:
  finalizers:
  - finalizer.stable.example.com
```

The first delete request on an object with finalizers sets a value for the **metadata.deletionTimestamp** field instead of deleting the object. This triggers controllers watching the object to execute any finalizers they handle.

Each controller then removes the finalizer from the list and issues the delete request again. This request deletes the object only if the list of finalizers is empty, meaning all finalizers are done.

第20章 ガベージコレクション

20.1. 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。

- Container garbage collection: Removes terminated containers.
- Image garbage collection: Removes images not referenced by any running pods.

20.2. 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** などの単位のサフィックスを使用して指定することができます。
maximum-dead-containers-per-container	The number of instances to retain per pod container. The default is **1**.
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。

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。

You can specify values for these settings in the **kubeletArguments** section of the **/etc/origin/node/node-config.yaml** file on node hosts. Add the section if it does not already exist:

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"
```

20.2.1. 削除するコンテナーの検出

ガベージコレクターの各ループでは、以下の手順が実行されます。

1. Retrieve a list of available containers.
2. Filter out all containers that are running or are not alive longer than the **minimum-container-ttl-duration** parameter.
3. Classify all remaining containers into equivalence classes based on pod and image name membership.
4. Remove all unidentified containers (containers that are managed by kubelet but their name is malformed).
5. For each class that contains more containers than the **maximum-dead-containers-per-container** parameter, sort containers in the class by creation time.
6. Start removing containers from the oldest first until the **maximum-dead-containers-per-container** parameter is met.
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. If this is still not enough, sort all containers in the list and start removing containers from the oldest first until the **maximum-dead-containers** criterion is met.

重要

各種のニーズに合わせてデフォルト設定を更新してください。

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。

20.3. 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	The percent of disk usage (expressed as an integer) which triggers image garbage collection. The default is **85**.
image-gc-low-threshold	The percent of disk usage (expressed as an integer) to which image garbage collection attempts to free. Default is **80**.

You can specify values for these settings in the **kubeletArguments** section of the **/etc/origin/node/node-config.yaml** file on node hosts. Add the section if it does not already exist:

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"
```

20.3.1. 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21章 ノードリソースの割り当て

21.1. 概要

To provide more reliable scheduling and minimize node resource overcommitment, each node can reserve a portion of its resources for use by all underlying node components (e.g., kubelet, kube-proxy, Docker) and the remaining system components (e.g., **sshd**, **NetworkManager**) on the host. Once specified, the scheduler has more information about the resources (e.g., memory, CPU) a node has allocated for pods.

21.2. 割り当てられるリソースについてのノードの設定

Resources reserved for node components are based on two node settings:

設定	説明
kube-reserved	Resources reserved for node components. Default is none.
system-reserved	Resources reserved for the remaining system components. Default is none.

You can set these in the **kubeletArguments** section of the node configuration file (the **/etc/origin/node/node-config.yaml** file by default) using a set of **<resource_type>=<resource_quantity>** pairs (e.g., **cpu=200m,memory=512Mi**). Add the section if it does not already exist:

例21.1 Node Allocatable Resources Settings

```
kubeletArguments:
  kube-reserved:
    - "cpu=200m,memory=512Mi"
  system-reserved:
    - "cpu=200m,memory=512Mi"
```

Currently, the **cpu** and **memory** resource types are supported. For **cpu**, the resource quantity is specified in units of cores (e.g., 200m, 0.5, 1). For **memory**, it is specified in units of bytes (e.g., 200Ki, 50Mi, 5Gi).

See Compute Resources for more details.

If a flag is not set, it defaults to **0**. If none of the flags are set, the allocated resource is set to the node's capacity as it was before the introduction of allocatable resources.

21.3. 割り当てられるリソースの計算

リソースの割り当てられる量は以下の数式に基づいて計算されます。

```
[Allocatable] = [Node Capacity] - [kube-reserved] - [system-reserved] - [Hard-Eviction-Thresholds]
```

注記

The withholding of **Hard-Eviction-Thresholds** from allocatable is a change in behavior to improve system reliability now that allocatable is enforced for end-user pods at the node level. The **experimental-allocatable-ignore-eviction** setting is available to preserve legacy behavior, but it will be deprecated in a future release.

If **[Allocatable]** is negative, it is set to 0.

21.4. VIEWING NODE ALLOCATABLE RESOURCES AND CAPACITY

To see a node's current capacity and allocatable resources, you can run:

```
$ oc get node/<node_name> -o yaml
...
status:
...
  allocatable:
    cpu: "4"
    memory: 8010948Ki
    pods: "110"
  capacity:
    cpu: "4"
    memory: 8010948Ki
    pods: "110"
...
```

21.5. ノードによって報告されるシステムリソース

Starting with OpenShift Container Platform 3.3, each node reports system resources utilized by the container runtime and kubelet. To better aid your ability to configure **--system-reserved** and **--kube-reserved**, you can introspect corresponding node's resource usage using the node summary API, which is accessible at **<master>/api/v1/nodes/<node>/proxy/stats/summary**.

For instance, to access the resources from **cluster.node22** node, you can run:

```
$ curl <certificate details> https://<master>/api/v1/nodes/cluster.node22/proxy/stats/summary
{
    "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"
         }
      ]
   }
}
```


See REST API Overview for more details about certificate details.

21.6. NODE ENFORCEMENT

The node is able to limit the total amount of resources that pods may consume based on the configured allocatable value. This feature significantly improves the reliability of the node by preventing pods from starving system services (for example: container runtime, node agent, etc.) for resources. It is strongly encouraged that administrators reserve resources based on the desired node utilization target in order to improve node reliability.

The node enforces resource constraints using a new cgroup hierarchy that enforces quality of service. All pods are launched in a dedicated cgroup hierarchy separate from system daemons.

To configure this ability, the following kubelet arguments are provided.

例21.2 ノードの cgroup 設定

```
kubeletArguments:
  cgroups-per-qos:
    - "true" 1
  cgroup-driver:
    - "systemd" 2
  enforce-node-allocatable:
    - "pods" 3
```

1 1 Enable or disable the new cgroup hierarchy managed by the node. Any change of this setting requires a full drain of the node. This flag must be true to allow the node to enforce node allocatable. We do not recommend users change this value.

2 2 The cgroup driver used by the node when managing cgroup hierarchies. This value must match the driver associated with the container runtime. Valid values are **systemd** and **cgroupfs**. The default is **systemd**.

3 A comma-delimited list of scopes for where the node should enforce node resource constraints. Valid values are **pods**, **system-reserved**, and **kube-reserved**. The default is **pods**. We do not recommend users change this value.

Optionally, the node can be made to enforce kube-reserved and system-reserved by specifying those

tokens in the enforce-node-allocatable flag. If specified, the corresponding **--kube-reserved-cgroup** or **--system-reserved-cgroup** needs to be provided. In future releases, the node and container runtime will be packaged in a common cgroup separate from **system.slice**. Until that time, we do not recommend users change the default value of enforce-node-allocatable flag.

Administrators should treat system daemons similar to Guaranteed pods. System daemons can burst within their bounding control groups and this behavior needs to be managed as part of cluster deployments. Enforcing system-reserved limits can lead to critical system services being CPU starved or OOM killed on the node. The recommendation is to enforce system-reserved only if operators have profiled their nodes exhaustively to determine precise estimates and are confident in their ability to recover if any process in that group is OOM killed.

As a result, we strongly recommended that users only enforce node allocatable for **pods** by default, and set aside appropriate reservations for system daemons to maintain overall node reliability.

21.7. エビクションしきい値

If a node is under memory pressure, it can impact the entire node and all pods running on it. If a system daemon is using more than its reserved amount of memory, an OOM event may occur that can impact the entire node and all pods running on it. To avoid (or reduce the probability of) system OOMs the node provides Out Of Resource Handling.

By reserving some memory via the **--eviction-hard** flag, the node attempts to evict pods whenever memory availability on the node drops below the absolute value or percentage. If system daemons did not exist on a node, pods are limited to the memory **capacity - eviction-hard**. For this reason, resources set aside as a buffer for eviction before reaching out of memory conditions are not available for pods.

Here is an example to illustrate the impact of node allocatable for memory:

- Node capacity is **32Gi**
- --kube-reserved is **2Gi**
- --system-reserved is **1Gi**
- --eviction-hard is set to **<100Mi**.

For this node, the effective node allocatable value is **28.9Gi**. If the node and system components use up all their reservation, the memory available for pods is **28.9Gi**, and kubelet will evict pods when it exceeds this usage.

If we enforce node allocatable (**28.9Gi**) via top level cgroups, then pods can never exceed **28.9Gi**. Evictions would not be performed unless system daemons are consuming more than **3.1Gi** of memory.

If system daemons do not use up all their reservation, with the above example, pods would face memcg OOM kills from their bounding cgroup before node evictions kick in. To better enforce QoS under this situation, the node applies the hard eviction thresholds to the top-level cgroup for all pods to be **Node Allocatable + Eviction Hard Thresholds**.

If system daemons do not use up all their reservation, the node will evict pods whenever they consume more than **28.9Gi** of memory. If eviction does not occur in time, a pod will be OOM killed if pods consume **29Gi** of memory.

21.8. SCHEDULER

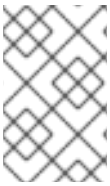
The scheduler now uses the value of **node.Status.Allocatable** instead of **node.Status.Capacity** to decide if a node will become a candidate for pod scheduling.

By default, the node will report its machine capacity as fully schedulable by the cluster.

第22章 OPAQUE INTEGER RESOURCES

22.1. 概要

Opaque integer resources allow cluster operators to provide new node-level resources that would be otherwise unknown to the system. Users can consume these resources in pod specifications, similar to CPU and memory. The scheduler performs resource accounting so that no more than the available amount is simultaneously allocated to pods.



注記

Opaque integer resources are Alpha currently, and only resource accounting is implemented. There is no resource quota or limit range support for these resources, and they have no impact on QoS.

Opaque integer resources are called **opaque** because OpenShift Container Platform does not know what the resource is, but will schedule a pod on a node only if enough of that resource is available. They are called **integer resources** because they must be available, or **advertised**, in integer amounts. The API server restricts quantities of these resources to whole numbers. Examples of **valid** quantities are **3**, **3000m**, and **3Ki**.

Opaque integer resources can be used to allocate:

- Last-level cache (LLC)
- Graphics processing unit (GPU) devices
- Field-programmable gate array (FPGA) devices
- Slots for sharing bandwidth to a parallel file system.

For example, if a node has 800 GiB of a special kind of disk storage, you could create a name for the special storage, such as **opaque-int-resource-special-storage**. You could advertise it in chunks of a certain size, such as 100 GiB. In that case, your node would advertise that it has eight resources of type **opaque-int-resource-special-storage**.

Opaque integer resource names must begin with the prefix **pod.alpha.kubernetes.io/opaque-int-resource-**.

22.2. CREATING OPAQUE INTEGER RESOURCES

There are two steps required to use opaque integer resources. First, the cluster operator must name and advertise a per-node opaque resource on one or more nodes. Second, application developer must request the opaque resource in pods.

To make opaque integer resources available:

1. Allocate the resource and assign a name starting with **pod.alpha.kubernetes.io/opaque-int-resource-**
2. Advertise a new opaque integer resource by submitting a PATCH HTTP request to the API server that specifies the available quantity in the **status.capacity** for a node in the cluster. For example, the following HTTP request advertises five **foo** resources on the **openshift-node-1** node.

```
PATCH /api/v1/nodes/openshift-node-1/status HTTP/1.1
Accept: application/json
Content-Type: application/json-patch+json
Host: openshift-master:8080

[
  {
    "op": "add",
    "path": "/status/capacity/pod.alpha.kubernetes.io~1opaque-int-resource-foo",
    "value": "5"
  }
]
```

注記

The **~1** in the **path** is the encoding for the character /. The operation path value in the JSON-Patch is interpreted as a JSON-Pointer. For more details, refer to IETF RFC 6901, section 3 .

After this operation, the node **status.capacity** includes a new resource. The **status.allocatable** field is updated automatically with the new resource asynchronously.

注記

Since the scheduler uses the node **status.allocatable** value when evaluating pod fitness, there might be a short delay between patching the node capacity with a new resource and the first pod that requests the resource to be scheduled on that node.

The application developer can then consume the opaque resources by editing the pod config to include the name of the opaque resource as a key in the **spec.containers[].resources.requests** field.

For example: The following pod requests two CPUs and one **foo** (an opaque resource).

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  containers:
  - name: my-container
    image: myimage
    resources:
      requests:
        cpu: 2
        pod.alpha.kubernetes.io/opaque-int-resource-foo: 1
```

The pod will be scheduled only if all of the resource requests are satisfied (including CPU, memory, and any opaque resources). The pod will remain in the **PENDING** state while the resource request cannot be met by any node.

```
Conditions:
  Type    Status
```

```
  PodScheduled  False
...
Events:
  FirstSeen  LastSeen Count From    SubObjectPath Type   Reason     Message
  ---------  -------- ----- ----    ------------- -------- ------     -------
  14s      0s  6 default-scheduler  Warning   FailedScheduling  No nodes are available that match all of
the following predicates:: Insufficient pod.alpha.kubernetes.io/opaque-int-resource-foo (1).
```

This information can also be found in the Developer Guide under Quotas and Limit Ranges.

第23章 オーバーコミット

23.1. 概要

Containers can specify compute resource requests and limits . Requests are used for scheduling your container and provide a minimum service guarantee. Limits constrain the amount of compute resource that may be consumed on your node.

scheduler 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`
```

The Fluentd **buffer_queue_limit** is 32.

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 実施されます。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

A container is guaranteed the amount of memory it requests. A container may use more memory than requested, but once it exceeds its requested amount, it could be killed in a low memory situation on the node.

If a container uses less memory than requested, it will not be killed unless system tasks or daemons need more memory than was accounted for in the node's resource reservation. If a container specifies a limit on memory, it is immediately killed if it exceeds the limit amount.

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。

優先順位	クラス名	説明
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。

Memory is an incompressible resource, so in low memory situations, containers that have the lowest priority are killed first:

- **Guaranteed** containers are considered top priority, and are guaranteed to only be killed if they exceed their limits, or if the system is under memory pressure and there are no lower priority containers that can be evicted.
- **Burstable** containers under system memory pressure are more likely to be killed once they exceed their requests and no other **BestEffort** containers exist.
- **BestEffort** containers are treated with the lowest priority. Processes in these containers are first to be killed if the system runs out of memory.

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。(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) デフォルトの制限で 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。

To configure **experimental-qos-reserved**, edit the **/etc/origin/node/node-config.yaml** file for the node.

```
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

Nodes by default enforce specified CPU limits using the CPU CFS quota support in the Linux kernel. If you do not want to enforce CPU limits on the node, you can disable its enforcement by modifying the node configuration file (the **node-config.yaml** file) to include the following:

```
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。
- 2 つ目の方法として **resource-reserver** Pod を作成できます。この Pod は、クラスターによるスケジュールの対象外となるようノードで容量を確保します。以下は例になります。

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。

Additionally, the node server needs to be configured to read the definition from the node configuration directory, by naming the directory in the **kubeletArguments.config** field of the node configuration file (usually named **node-config.yaml**):

```
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

You can disable swap by default on your nodes in order to preserve quality of service guarantees. Otherwise, physical resources on a node can oversubscribe, affecting the resource guarantees the Kubernetes scheduler makes during pod placement.

For example, if two guaranteed pods have reached their memory limit, each container could start using swap memory. Eventually, if there is not enough swap space, processes in the pods can be terminated due to the system being oversubscribed.

To disable swap:

```
$ swapoff -a
```

Failing to disable swap results in nodes not recognizing that they are experiencing **MemoryPressure**, resulting in pods not receiving the memory they made in their scheduling request. As a result, additional pods are placed on the node to further increase memory pressure, ultimately increasing your risk of experiencing a system out of memory (OOM) event.

重要

If swap is enabled, any out of resource handling eviction thresholds for available memory will not work as expected. Take advantage of out of resource handling to allow pods to be evicted from a node when it is under memory pressure, and rescheduled on an alternative node that has no such pressure.

第24章 INGRESS トラフィックの固有の外部 IP の割り当て

24.1. 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 uses by the Loadbalancer when choosing an external IP address for the service.
- 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ースコントローラー (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。

24.2. 制限

ExternalIP を使用するには、以下を実行できます。

- Select an IP address from the **externalIPNetworkCIDRs** range.
- Have an IP address assigned from the **ingressIPNetworkCIDR** pool in the master configuration file. In this case, OpenShift Container Platform implements a non-cloud version of the load balancer service type and assigns IP addresses to the services.

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。以下は例になります。

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。

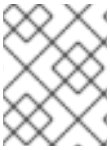
ただし、Ingress コントローラーが外部 IP を割り当てる場合、ポートのクラッシュは問題とはなりません。コントローラーが各サービスに固有のアドレスを割り当てるためです。

24.3. 固有の外部 IP を使用するようクラスターを設定する

In non-cloud clusters, **ingressIPNetworkCIDR** is set by default to **172.29.0.0/16**. If your cluster environment is not already using this private range, you can use the default. However, if you want to use a different range, then you must set **ingressIPNetworkCIDR** in the **/etc/origin/master/master-config.yaml** file before you assign an ingress IP. Then, restart the master service.

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 アドレスより少なくなければなりません。

/etc/origin/master/master-config.yaml のサンプル

```
networkConfig:
  ingressIPNetworkCIDR: 172.29.0.0/16
```

24.3.1. 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
```

24.4. 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。以下は例になります。

```
# 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。

24.4.1. 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。

```
# systemctl restart atomic-openshift-master-api atomic-openshift-master-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 (これは内部 IP アドレス一覧に追加される一覧です)。

第25章 OUT OF RESOURCE (リソース不足) エラーの処理

25.1. 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、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モニターし、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、シナリオ例や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

If swap memory is enabled for a node, that node cannot detect that it is under **MemoryPressure**.

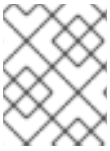
メモリーベースのエビクションを利用するには、オペレーターは swap を無効にする必要があります。

25.2. 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。

An eviciton policy is a combination of an eviction trigger signal with a specific eviction threshold value, that is set in the node configuration file or through the command line. Evictions can be either hard, where a node takes immediate action on a pod that exceeds a threshold, or soft, where a node allows a grace period before taking action. See the sections below for important information the differences between hard and soft evictions.

By using well-configured eviction policies, a node can proactively monitor for and prevent against total starvation of a compute resource.



注記

When the node fails a pod, it terminates all containers in the pod, and the **PodPhase** is transitioned to **Failed**.

25.2.1. ノード設定を使用したポリシーの作成

To configure an eviction policy, edit the node configuration file (the **/etc/origin/node/node-config.yaml** file) to specify the eviction thresholds under the **eviction-hard** or **eviction-soft** parameters.

例:

例25.1 Sample Node Configuration file for a hard eviction

```
kubeletArguments:
  eviction-hard: 1
  - memory.available<500Mi 2
  - nodefs.available<500Mi
  - nodefs.inodesFree<100Mi
  - imagefs.available<100Mi
  - imagefs.inodesFree<100Mi
```

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。

例25.2 Sample Node Configuration file for a soft eviction

```
kubeletArguments:
  eviction-soft: 1
  - memory.available<500Mi 2
  - nodefs.available<500Mi
  - nodefs.inodesFree<100Mi
  - imagefs.available<100Mi
  - imagefs.inodesFree<100Mi
  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。

1. 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
```

25.2.2. 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。エビクションシグナルは、しきい値と共にエビクションのしきい値に追加できます。

The value of each signal is described in the **Description** column based on the node summary API.

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25.1 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**	Available diskspace on either the node root file system or image file system has exceeded an eviction threshold.
	nodefs.inodesFree	**nodefs.inodesFree** = **node.stats.fs.inodesFree**	

ノードの状態	エビクションシグナル	値	説明
	imagefs.available	**imagefs.available** = **node.stats.runtime.imagefs.available**	
	imagefs.inodesFree	**imagefs.inodesFree** = **node.stats.runtime.imagefs.inodesFree**	

Each of the above signals supports either a literal or percentage-based value. The percentage-based value is calculated relative to the total capacity associated with each signal.

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。

The node supports the **nodefs** and **imagefs** file system partitions when detecting disk pressure, as follows:

- The **nodefs** file system that the node uses for local disk volumes, daemon logs, and so on (for example, the file system that provides /).
- The **imagefs** file system that the container runtime uses for storing images and individual container writable layers.

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。

If you store volumes and logs in a dedicated file system, the node will not monitor that file system.

注記

As of OpenShift Container Platform 3.4, the node supports the ability to trigger eviction decisions based on disk pressure. Before evicting pods becuase of disk pressure, the node also performs container and image garbage collection. In future releases, garbage collection will be deprecated in favor of a pure disk-eviction based configuration.

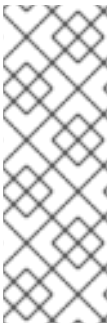
25.2.3. エビクションのしきい値について

You can configure a node to specify eviction thresholds, which triggers the node to reclaim resources, by adding a threshold to the node configuration file.

If an eviction threshold is met, independent of its associated grace period, the node reports a condition indicating that the node is under memory or disk pressure. This prevents the scheduler from scheduling any additional pods on the node while attempts to reclaim resources are made.

The node continues to report node status updates at the frequency specified by the **node-status-update-frequency** argument, which defaults to **10s** (ten seconds).

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

Soft eviction usage is more common when you are targeting a certain level of utilization, but can tolerate temporary spikes. We recommended setting the soft eviction threshold lower than the hard eviction threshold, but the time period can be operator-specific. The system reservation should also cover the soft eviction threshold.

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>
```

- the **eviction-signal** value can be any supported eviction signal.
- the **operator** value is **<**.
- the **quantity** value must match the quantity representation used by Kubernetes and can be expressed as a percentage if it ends with the **%** token.

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。

25.2.3.1. ハードエビクションのしきい値について

A hard eviction threshold has no grace period and, if observed, the node takes immediate action to reclaim the associated starved resource. If a hard eviction threshold is met, the node kills the pod immediately with no graceful termination.

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。

Sample Node Configuration file with hard eviction thresholds

```
kubeletArguments:
  eviction-hard:
  - memory.available<500Mi
  - nodefs.available<500Mi
  - nodefs.inodesFree<100Mi
  - imagefs.available<100Mi
  - imagefs.inodesFree<100Mi
```

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。

25.2.3.1.1. 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** の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%
...
```

25.2.3.2. ソフトエビクションのしきい値について

A soft eviction threshold pairs an eviction threshold with a required administrator-specified grace period. The node does not reclaim resources associated with the eviction signal until that grace period is exceeded. If no grace period is provided in the node configuration the node errors on startup.

In addition, if a soft eviction threshold is met, an operator can specify a maximum allowed pod termination grace period to use when evicting pods from the node. If **eviction-max-pod-grace-period** is specified, the node uses the lesser value among the **pod.Spec.TerminationGracePeriodSeconds** and the maximum-allowed grace period. If not specified, the node kills pods immediately with no graceful termination.

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**: a set of eviction thresholds (for example, **memory.available<1.5Gi**) that, if met over a corresponding grace period, triggers a pod eviction.
- **eviction-soft-grace-period**: a set of eviction grace periods (for example, **memory.available=1m30s**) that correspond to how long a soft eviction threshold must hold before triggering a pod eviction.

- **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。

Sample Node Configuration files with soft eviction thresholds

```
kubeletArguments:
  eviction-soft:
  - memory.available<500Mi
  - nodefs.available<500Mi
  - nodefs.inodesFree<100Mi
  - imagefs.available<100Mi
  - imagefs.inodesFree<100Mi
  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。

25.3. 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。

Set **system-reserved** equal to the amount of resource you want available to the scheduler for deploying pods and for system-daemons. Evictions should only occur if pods use more than their requested amount of an allocatable resource.

ノードは 2 つの値を報告します。

- **Capacity**: How much resource is on the machine
- **Allocatable**: スケジューリング用に利用可能にされるリソースの量です。

To configure the amount of allocatable resources, edit the node configuration file (the **/etc/origin/node/node-config.yaml** file) to add or modify the **system-reserved** parameter for **eviction-hard** or **eviction-soft**.

+

```
kubeletArguments:
  eviction-hard: 1
    - "memory.available<500Mi"
  system-reserved:
    - "1.5Gi"
```

1 このしきい値は、**eviction-hard** または **eviction-soft** のいずれかにできます。

1. 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
```

25.4. ノードの状態変動の制御

If a node is oscillating above and below a soft eviction threshold, but not exceeding its associated grace period, the corresponding node condition oscillates between **true** and **false**, which can cause problems for the scheduler.

To prevent this oscillation, set the **eviction-pressure-transition-period** parameter to control how long the node must wait before transitioning out of a pressure condition.

1. Edit or add the parameter to the **kubeletArguments** section of the node configuration file (the **/etc/origin/node/node-config.yaml**) using a set of **<resource_type>=<resource_quantity>** pairs.

```
kubeletArguments:
  eviction-pressure-transition-period="5m"
```

+ The node toggles the condition back to **false** when the node has not observed an eviction threshold being met for the specified pressure condition for the specified period.

+

注記

Use the default value (5 minutes) before doing any adjustments. The default choice is intended to allow the system to stabilize, and to prevent the scheduler from assigning new pods to the node before it has settled.

1. 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
```

25.5. ノードレベルのリソースの回収

If an eviction criteria is satisfied, the node initiates the process of reclaiming the pressured resource until the signal goes below the defined threshold. During this time, the node does not support scheduling any new pods.

The node attempts to reclaim node-level resources prior to evicting end-user pods, based on whether the host system has a dedicated **imagefs** configured for the container runtime.

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:

- If the **nodefs** file system meets eviction thresholds, the node frees up disk space in the following order:
  - Delete dead pods/containers
- If the **imagefs** file system meets eviction thresholds, the node frees up disk space in the following order:
  - Delete all unused images

Imagefs が設定されていない場合
ホストシステムに **imagefs** がされていない場合:

- If the **nodefs** file system meets eviction thresholds, the node frees up disk space in the following order:
  - Delete dead pods/containers
  - Delete all unused images

25.6. POD エビクションについて

If an eviction threshold is met and the grace period is passed, the node initiates the process of evicting pods until the signal goes below the defined threshold.

The node ranks pods for eviction by their quality of service, and, among those with the same quality of service, by the consumption of the starved compute resource relative to the pod's scheduling request.

Each QOS level has an OOM score, which the Linux out-of-memory tool (OOM killer) uses to determine which pods to kill. See Understanding Quality of Service and Out of Memory Killer below.

The following table lists each QOS level and the associated OOM score.

表25.2 Quality of Service (QoS) レベル

QoS (Quality of Service)	説明
Guaranteed	Pods that consume the highest amount of the starved resource relative to their request are failed first. If no pod has exceeded its request, the strategy targets the largest consumer of the starved resource.
Burstable	Pods that consume the highest amount of the starved resource relative to their request for that resource are failed first. If no pod has exceeded its request, the strategy targets the largest consumer of the starved resource.
BestEffort	Pods that consume the highest amount of the starved resource are failed first.

A **Guaranteed** pod will never be evicted because of another pod's resource consumption unless a system daemon (such as node, **docker**, **journald**) is consuming more resources than were reserved using **system-reserved**, or **kube-reserved** allocations or if the node has only **Guaranteed** pods remaining.

If the node has only **Guaranteed** pods remaining, the node evicts a **Guaranteed** pod that least impacts node stability and limits the impact of the unexpected consumption to other **Guaranteed** pods.

Local disk is a **BestEffort** resource. If necessary, the node evicts pods one at a time to reclaim disk when **DiskPressure** is encountered. The node ranks pods by quality of service. If the node is responding to inode starvation, it will reclaim inodes by evicting pods with the lowest quality of service first. If the node is responding to lack of available disk, it will rank pods within a quality of service that consumes the largest amount of local disk, and evict those pods first.

25.6.1. QoS および Out of Memory Killer について

If the node experiences a system out of memory (OOM) event before it is able to reclaim memory, the node depends on the OOM killer to respond.

The node sets a **oom_score_adj** value for each container based on the quality of service for the pod.

表25.3 Quality of Service (QoS) レベル

QoS (Quality of Service)	oom_score_adj 値
Guaranteed	-998
Burstable	min(max(2, 1000 - (1000 * memoryRequestBytes) / machineMemoryCapacityBytes), 999)
BestEffort	1000

If the node is unable to reclaim memory prior to experiencing a system OOM event, the **oom_killer** calculates an **oom_score**:

```
% of node memory a container is using + `oom_score_adj` = `oom_score`
```

The node then kills the container with the highest score.

Containers with the lowest quality of service that are consuming the largest amount of memory relative to the scheduling request are failed first.

Unlike pod eviction, if a pod container is OOM failed, it can be restarted by the node based on the node restart policy.

25.7. POD スケジューラーおよび OOR 状態について

The scheduler views node conditions when placing additional pods on the node. For example, if the node has an eviction threshold like the following:

```
eviction-hard is "memory.available<500Mi"
```

and available memory falls below 500Mi, the node reports a value in **Node.Status.Conditions** as **MemoryPressure** as true.

表25.4 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	If a node reports this condition, the scheduler will not place **BestEffort** pods on that node.
DiskPressure	If a node reports this condition, the scheduler will not place any additional pods on that node.

25.8. シナリオ例

Consider the following scenario.

An opertator:

- has a node with a memory capacity of **10Gi**;
- wants to reserve 10% of memory capacity for system daemons (kernel, node, etc.);
- wants to evict pods at 95% memory utilization to reduce thrashing and incidence of system OOM.

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。

To reach that capacity, either some pod is using more than its request, or the system is using more than **1Gi**.

If a node has 10 Gi of capacity, and you want to reserve 10% of that capacity for the system daemons (**system-reserved**), perform the following calculation:

```
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。

If you want to turn on eviction so that eviction is triggered when the node observes that available memory falls below 10% of capacity for 30 seconds, or immediately when it falls below 5% of capacity, you need the scheduler to see allocatable as 8Gi. Therefore, ensure your system reservation covers the greater of your eviction thresholds.

```
capacity = 10 Gi
eviction-threshold = 10 Gi * .1 = 1 Gi
system-reserved = (10Gi * .1) + eviction-threshold = 2 Gi
allocatable = capacity - system-reserved = 8 Gi
```

Enter the following in the **node-config.yaml**:

```
kubeletArguments:
  system-reserved:
  - "2Gi"
  eviction-hard:
  - memory.available<.5Gi
  eviction-soft:
  - memory.available<1Gi
  eviction-soft-grace-period:
  - memory.available=30s
```

This configuration ensures that the scheduler does not place pods on a node that immediately induce memory pressure and trigger eviction assuming those pods use less than their configured request.

25.9. 推奨される対策

25.9.1. DaemonSets and Out of Resource Handling

If a node evicts a pod that was created by a DaemonSet, the pod will immediately be recreated and rescheduled back to the same node, because the node has no ability to distinguish a pod created from a DaemonSet versus any other object.

In general, DaemonSets should not create **BestEffort** pods to avoid being identified as a candidate pod for eviction. Instead DaemonSets should ideally launch **Guaranteed** pods.

第26章 ルーターのモニタリングおよびデバッグ

26.1. 概要

Depending on the underlying implementation, you can monitor a running router in multiple ways. This topic discusses the HAProxy template router and the components to check to ensure its health.

26.2. 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。

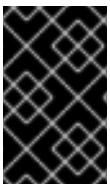
26.3. 統計ビューの無効化

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理由により **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。

26.4. 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26.1 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。

環境変数	説明
ROUTER_LOG_LEVEL	これはオプションであり、HAProxy ログレベルを変更する際に設定します。設定されていない場合は、デフォルトのログレベルは **warning** 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	これはオプションであり、カスタマイズされた HAProxy ログ形式を定義する際に設定されます。これを 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
```

26.5. 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27章 高可用性

27.1. 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の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、パーマンスは依然として影響を受けます。**keepalived** はそれぞれの VIP が設定内のノードによって提供されるようにし、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ことがあります。ipfailover が複数の VIP を同じノードに配置する場合、外部から一連の VIP 間で負荷分散を行う方法は失敗する可能性があります。

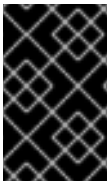
ingressIP を使用する場合は、ipfailover を ingressIP 範囲と同じ VIP 範囲を持つように設定できます。また、モニターポートを無効にすることもできます。この場合、すべての VIP がクラスター内の同じノードに表示されます。すべてのユーザーが ingressIP でサービスをセットアップし、これを高い可用性のあるサービスにすることができます。

重要

クラスター内の VIP の最大数は 255 です。

27.2. 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) で OpenShift Container Platform の高可用性を維持することができます。

As an administrator, you can configure ipfailover on an entire cluster, or on a subset of nodes, as defined by the label selector. You can also configure multiple IP failover deployment configurations in your cluster, where each one is independent of the others. The **oc adm ipfailover** command creates an ipfailover deployment configuration which ensures that a failover pod runs on each of the nodes matching the constraints or the label used. This pod runs Keepalived which uses VRRP (Virtual Router Redundancy Protocol) among all the **Keepalived** daemons to ensure that the service on the watched port is available, and if it is not, **Keepalived** will automatically float the VIPs.

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
```

27.2.1. 仮想 IP アドレス

Keepalived manages a set of virtual IP addresses. The administrator must make sure that all these addresses:

- 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。

27.2.2. 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。

When a check script is not provided, a simple default script is run that tests the TCP connection. This default test is suppressed when the monitor port is **0**.

For each VIP, **keepalived** keeps the state of the node. The VIP on the node may be in **MASTER**, **BACKUP**, or **FAULT** state. All VIPs on the node that are not in the **FAULT** state participate in the negotiation to decide which will be **MASTER** for the VIP. All of the losers enter the **BACKUP** state. When the **check** script on the **MASTER** fails, the VIP enters the **FAULT** state and triggers a renegotiation. When the **BACKUP** fails, the VIP enters the **FAULT** state. When the **check** script passes again on a VIP in the **FAULT** state, it exits **FAULT** and negotiates for **MASTER**. The resulting state is either **MASTER** or **BACKUP**.

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。

These scripts run in the IP failover pod and use the pod's file system, not the host file system. The options require the full path to the script. The administrator must make the script available in the pod to extract the results from running the **notify** script. The recommended approach for providing the scripts is to use a ConfigMap.

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 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. 変更を保存して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。

27.2.3. 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
```

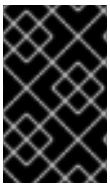
c. **oc edit dc ipf-ha-router** を使用してルーターデプロイメント設定を編集します。

```
...
   spec:
     containers:
     - env:
```

```
        - name: OPENSHIFT_HA_PREEMPTION (1)
          value: nopreempt
...
```

27.2.4. 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。

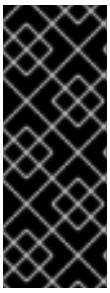
keepalived デーモンを起動する前に、起動スクリプトは、マルチキャストトラフィックのフローを許可する **iptables** ルールを検証します。このルールがない場合、起動スクリプトは新規ルールを作成し、これを IP テーブル接続に追加します。この新規ルールが IP テーブルに追加される場所は **--iptables-chain=** オプションによって異なります。**--iptables-chain=** オプションが指定される場合、ルールはオプションで指定されるチェーンに追加されます。そうでない場合は、ルールは **INPUT** チェーンに追加されます。

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) が設定されます。

```
$ 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;
```

27.2.5. コマンドラインオプションおよび環境変数

表27.1 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	備考
--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 のインターフェース名で、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です。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.)。詳細については、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。
--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。
--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。

オプション	変数名	デフォルト	備考
--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。

27.2.6. VRRP ID オフセット

Each ipfailover pod managed by the ipfailover deployment configuration (1 pod per node/replica) runs a **keepalived** daemon. As more ipfailover deployment configurations are configured, more pods are created and more daemons join into the common VRRP negotiation. This negotiation is done by all the **keepalived** daemons and it determines which nodes will service which VIPs.

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。

27.2.7. 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、地理的区分 US west 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"
   ```

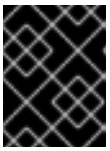
2. サービスアカウントを作成します。ipfailover を使用したり、(環境ポリシーによって異なる) ルーターを使用する場合は事前に作成された **router** サービスアカウントか、または新規の ipfailover サービスアカウントのいずれかを再利用できます。
   以下の例は、**デフォルト** namespace で ipfailover という名前の新規サービスアカウントを作成します。

   ```
   $ 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。



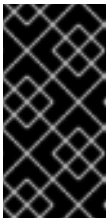
重要

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. Run the **geo-cache** service with a replica on each of the nodes. See an example configuration for running a **geo-cache** service.

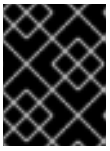


重要

Make sure that you replace the **myimages/geo-cache** Docker image referenced in the file with your intended image. Change the number of replicas to the number of nodes in the **geo-cache** label. Check that the label matches the one used in the first step.

```
$ 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** の値は、**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。

27.2.7.1. 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 デプロイするインスタンスの数を指定します。

ipfailover がデプロイされる場所を制限します。

モニターする仮想 IP アドレスです。

各ノード上の ipfailover がモニターするポートです。

27.2.8. 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。

- ローリングアップデート (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:
    recreateParams:
      timeoutSeconds: 600
    resources: {}
    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。

27.3. サービスの EXTERNALIP および NODEPORT の設定

The user can assign VIPs as ExternalIPs in a service. **Keepalived** makes sure that each VIP is served on some node in the ipfailover configuration. When a request arrives on the node, the service that is running on all nodes in the cluster, load balances the request among the service's endpoints.

The NodePorts can be set to the ipfailover watch port so that **keepalived** can check the application is running. The NodePort is exposed on all nodes in the cluster, therefore it is available to **keepalived** on all ipfailover nodes.

27.4. INGRESSIP の高可用性

In non-cloud clusters, ipfailover and ingressIP to a service can be combined. The result is high availability services for users that create services using ingressIP.

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28章 IPTABLES

28.1. 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。OpenShift Container Platform は問題をモニターし、解決しますが、Docker サービスはこれを実行しません。

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。

28.2. 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。

Independent of other use, such as a firewall, OpenShift Container Platform and the the Docker service manage chains in some of the tables. The chains are inserted in specific order and the rules are specific to their needs.

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。

28.3. 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
```

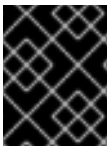
iptables.service を実行する必要がある場合、制限された設定を設定ファイルに維持し、OpenShift Container Platform および Docker を使用してそれらが必要とするルールをインストールするようにします。

iptables.service 設定は以下から読み取られます。

```
/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29章 ストラテジーによるビルドのセキュリティー保護

29.1. 概要

Builds in OpenShift Container Platform are run in privileged containers that have access to the Docker daemon socket. As a security measure, it is recommended to limit who can run builds and the strategy that is used for those builds. Custom builds are inherently less safe than Source builds, given that they can execute any code in the build with potentially full access to the node's Docker socket, and as such are disabled by default. Docker build permission should also be granted with caution as a vulnerability in the Docker build logic could result in a privileges being granted on the host node.

By default, all users that can create builds are granted permission to use the Docker and Source-to-Image build strategies. Users with **cluster-admin** privileges can enable the Custom build strategy, as referenced in the Restricting Build Strategies to a User Globally section of this page.

You can control who can build with what build strategy using an authorization policy. Each build strategy has a corresponding build subresource. A user must have permission to create a build **and** permission to create on the build strategy subresource in order to create builds using that strategy. Default roles are provided which grant the **create** permission on the build strategy subresource.

表29.1 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	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

29.2. ビルドストラテジーのグローバルな無効化

To prevent access to a particular build strategy globally, log in as a user with **cluster-admin** privileges and remove the corresponding role from the **system:authenticated** group:

```
$ oc adm policy remove-cluster-role-from-group system:build-strategy-custom system:authenticated
$ oc adm policy remove-cluster-role-from-group system:build-strategy-docker system:authenticated
$ oc adm policy remove-cluster-role-from-group system:build-strategy-source system:authenticated
$ oc adm policy remove-cluster-role-from-group system:build-strategy-jenkinspipeline
system:authenticated
```

In versions prior to 3.2, the build strategy subresources were included in the **admin** and **edit** roles. Ensure the build strategy subresources are also removed from these roles:

```
$ 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。

例29.1 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。

29.3. 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。

29.4. 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30章 SECCOMP を使用したアプリケーション機能の制限

30.1. 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有効になります。

- **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。

30.2. 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
```

30.3. 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. Configure your nodes to use the **seccomp-profile-root** where your profiles will be stored. In the **node-config.yaml** via the **kubeletArguments**:

```
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. In order to control which profiles may be used, and to set the default profile, configure your SCC via the **seccompProfiles** field. The first profile will be used as a default.
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。
For example, if you are using the default **docker/default** profile, configure the **restricted** SCC with:

```
seccompProfiles:
- docker/default
```

30.4. OPENSHIFT CONTAINER PLATFORM でのカスタム SECCOMP プロファイルの設定

To ensure pods in your cluster run with a custom profile in the **restricted** SCC:

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. Configure the **restricted** SCC:

```
seccompProfiles:
- localhost/<profile-name>
```

第31章 SYSCTL

31.1. 概要

Sysctl settings are exposed via Kubernetes, allowing users to modify certain kernel parameters at runtime for namespaces within a container. Only sysctls that are namespaced can be set independently on pods; if a sysctl is not namespaced (called **node-level**), it cannot be set within OpenShift Container Platform. Moreover, only those sysctls considered **safe** are whitelisted by default; other **unsafe** sysctls can be manually enabled on the node to be available to the user.

31.2. UNDERSTANDING SYSCTLS

In Linux, the sysctl interface allows an administrator to modify kernel parameters at runtime. Parameters are available via the **/proc/sys/** virtual process file system. The parameters cover various subsystems such as:

- カーネル (共通のプレフィックス: **kernel.**)
- ネットワーク (共通のプレフィックス: **net.**)
- 仮想メモリー (共通のプレフィックス: **vm.**)
- MDADM (共通のプレフィックス: **dev.**)

追加のサブシステムについては、カーネルのドキュメントで説明されています。す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
```

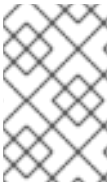
31.3. NAMESPACED VS NODE-LEVEL SYSCTLS

A number of sysctls are **namespaced** in today's Linux kernels. This means that they can be set independently for each pod on a node. Being namespaced is a requirement for sysctls to be accessible in a pod context within Kubernetes.

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.***
- **net.***

Sysctls that are not namespaced are called **node-level** and must be set manually by the cluster administrator, either by means of the underlying Linux distribution of the nodes (e.g., via **/etc/sysctls.conf**) or using a DaemonSet with privileged containers.



注記

Consider marking nodes with special sysctls as tainted. Only schedule pods onto them that need those sysctl settings. Use the Kubernetes **taints and toleration** feature to implement this.

31.4. SAFE VS UNSAFE SYSCTLS

Sysctls are grouped into **safe** and **unsafe** sysctls. In addition to proper namespacing, a safe sysctl must be properly isolated between pods on the same node. This means that setting a safe sysctl for one pod:

- must not have any influence on any other pod on the node,
- must not allow to harm the node's health, and
- must not allow to gain CPU or memory resources outside of the resource limits of a pod.

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。

For OpenShift Container Platform 3.3.1, the following sysctls are supported (whitelisted) in the safe set:

- **kernel.shm_rmid_forced**
- **net.ipv4.ip_local_port_range**

This list will be extended in future versions when the kubelet supports better isolation mechanisms.

All safe sysctls are enabled by default. All unsafe sysctls are disabled by default and must be allowed manually by the cluster administrator on a per-node basis. Pods with disabled unsafe sysctls will be scheduled, but will fail to launch.

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。

31.5. ENABLING UNSAFE SYSCTLS

With the warning above in mind, the cluster administrator can allow certain unsafe sysctls for very special situations, e.g., high-performance or real-time application tuning.

If you want to use unsafe sysctls, cluster administrators must enable them individually on nodes. Only namespaced sysctls can be enabled this way.

1. Use the **kubeletArguments** field in the **/etc/origin/node/node-config.yaml** file, as described in Configuring Node Resources, to set the desired unsafe sysctls:

```
kubeletArguments:
  experimental-allowed-unsafe-sysctls:
    - "kernel.msg*,net.ipv4.route.min_pmtu"
```

2. 変更を適用するためにノードサービスを再起動します。

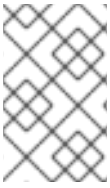
```
# systemctl restart atomic-openshift-node
```

31.6. SETTING SYSCTLS FOR A POD

Sysctls are set on pods using annotations. They apply to all containers in the same pod.

Here is an example, with different annotations for safe and unsafe sysctls:

```
apiVersion: v1
kind: Pod
metadata:
  name: sysctl-example
  annotations:
    security.alpha.kubernetes.io/sysctls: kernel.shm_rmid_forced=1
    security.alpha.kubernetes.io/unsafe-sysctls: net.ipv4.route.min_pmtu=1000,kernel.msgmax=1 2 3
spec:
  ...
```



注記

A pod with the unsafe sysctls specified above will fail to launch on any node that has not enabled those two unsafe sysctls explicitly. As with node-level sysctls, use the taints and toleration feature or labels on nodes to schedule those pods onto the right nodes.

第32章 データストア層でのデータの暗号化

32.1. 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。

32.2. 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
```

For more information about **master-config.yaml** and its format, see the Master Configuration Files topic.

32.3. 暗号化設定について

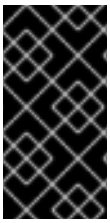
すべての利用可能なプロバイダーを含む暗号化設定ファイル

```
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」について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。

32.3.1. 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。

32.4. データの暗号化

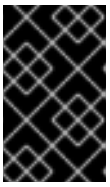
新規の暗号化設定ファイルを作成します。

```
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。

```
# systemctl restart atomic-openshift-master-api
```

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。

32.5. 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されます。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:** のプレフィックスが付けられます。

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。

32.6. 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。

32.7. 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。

```
# systemctl restart atomic-openshift-master-api
```

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。

```
# systemctl restart atomic-openshift-master-api
```

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。

32.8. 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。

```
# systemctl restart atomic-openshift-master-api
```

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。

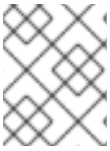
```
$ oc adm migrate storage --include=secrets --confirm
```

第33章 ENCRYPTING HOSTS WITH IPSEC

33.1. 概要

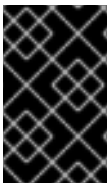
IPsec は、インターネットプロトコル (IP) を使用して通信するすべてのマスターとノードホスト間の通信を暗号化することによって OpenShift Container Platform クラスターのトラフィックを保護します。

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

This procedure should be repeated on each master host, then node host, in your cluster. Hosts that do not have IPsec enabled will not be able to communicate with a host that does.

33.2. ENCRYPTING HOSTS

33.2.1. 前提条件

- Ensure that **libreswan** 3.15 or later is installed on cluster hosts. If opportunistic group functionality is required, then **libreswan** version 3.19 or later is required.
- See the Configure the pod network on nodes section for information on how to configure the MTU to allow space for the IPSec header. This topic describes an IPSec configuration that requires 62 bytes. If the cluster is operating on an Ethernet network with an MTU of 1500 then the SDN MTU should be 1388, to allow for the overhead of IPSec and the SDN encapsulation. After modifying the MTU in the OpenShift Container Platform configuration, the SDN must be made aware of the change by removing the SDN interface and restarting the OpenShift Container Platform node process.

```
# systemctl stop atomic-openshift-node
# ovs-vsctl del-br br0
# systemctl start atomic-openshift-node
```

33.2.2. 証明書での IPsec の設定

By default, OpenShift Container Platform secures cluster management communication with mutually authenticated HTTPS communication. This means that both the client (for example, an OpenShift Container Platform node) and the server (for example, an OpenShift Container Platform api-server) send each other their certificates, which are checked against a known certificate authority (CA). These certificates are generated at cluster set up time and typically live on each host. These certificates can also be used to secure pod communications with IPsec.

This procedure assumes you have the following on each host:

- Cluster CA file
- Host client certificate file
- Host private key file

1. Determine what the certificate's nickname will be after it has been imported into the **libreswan** certificate database. The nickname is taken directly from the certificate's subject's Common Name (CN):

```
# openssl x509 \
  -in /path/to/client-certificate -subject -noout | \
  sed -n 's/.*CN=\(.*\)/\1/p'
```

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 /path/to/client-certificate \
  -inkey /path/to/private-key \
  -certfile /path/to/certificate-authority \
  -passout pass: \
  -out certs.p12
```

3. Import the **PKCS#12** file into the **libreswan** certificate database. The **-W** option is left empty because no password is assigned to the **PKCS#12** file, as it is only temporary.

```
# ipsec initnss
# pk12util -i certs.p12 -d sql:/etc/ipsec.d -W ""
# rm certs.p12
```

33.2.3. libreswan IPsec Policy

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。

If you are using **libreswan** 3.19 or later, then opportunistic group configuration is recommended. Otherwise, explicit connections are required.

33.2.3.1. Opportunistic Group Configuration

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

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. **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。このファイルでは、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設定されます。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 should contain the IP subnet of your cluster, which your hosts receive IP addresses from. By default, this causes all communication between hosts in the cluster subnet to be encrypted if the remote host's client certificate authenticates against the local host's Certificate Authority certificate. If the remote host's certificate does not authenticate, all traffic between the two hosts will be blocked.

   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。

33.2.3.2. Explicit Connection Configuration

In this configuration, each IPSec node configuration must explicitly list the configuration of every other node in the cluster. Using a configuration management tool such as Ansible to generate this file on each host is recommended.

1. This configuration also requires the full certificate subject of each node to be placed into the configuration for every other node. To read this subject from the node's certificate, use **openssl**:

   ```
   # openssl x509 \
     -in /path/to/client-certificate -text | \
   ```

```
    grep "Subject:" | \
    sed 's/[[:blank:]]*Subject: //'
```

2. 以下の行を、クラスター内のその他のノード用に各ノードの **/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
```

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。

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
```

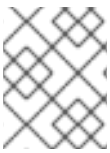
1 <this_node_cert_nickname> を手順 1 のノードの証明書ニックネームに置き換えます。

33.3. IPSEC FIREWALL CONFIGURATION

All nodes within the cluster need to allow IPSec related network traffic. This includes IP protocol numbers 50 and 51 as well as UDP port 500.

たとえば、クラスターノードがインターフェース **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

IPSec also uses UDP port 4500 for NAT traversal, though this should not apply to normal cluster deployments.

33.4. STARTING AND ENABLING IPSEC

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
```

33.5. OPTIMIZING IPSEC

See the Scaling and Performance Guide for performance suggestions when encrypting with IPSec.

33.6. 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34章 依存関係ツリーのビルド

34.1. 概要

OpenShift Container Platform uses image change triggers in a **BuildConfig** to detect when an image stream tag has been updated. You can use the **oc adm build-chain** command to build a dependency tree that identifies which images would be affected by updating an image in a specified image stream.

The **build-chain** tool can determine which builds to trigger; it analyzes the output of those builds to determine if they will in turn update another image stream tag. If they do, the tool continues to follow the dependency tree. Lastly, it outputs a graph specifying the image stream tags that would be impacted by an update to the top-level tag. The default output syntax for this tool is set to a human-readable format; the DOT format is also supported.

34.2. 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34.1 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35章 BACKUP AND RESTORE

35.1. 概要

In OpenShift Container Platform, you can **back up** (saving state to separate storage) and **restore** (recreating state from separate storage) at the cluster level. There is also some preliminary support for per-project backup. The full state of a cluster installation includes:

- etcd data on each master
- API objects
- registry storage
- volume storage

This topic does not cover how to back up and restore persistent storage, as those topics are left to the underlying storage provider. However, an example of how to perform a **generic** backup of application data is provided.

重要

This topic only provides a generic way of backing up applications and the OpenShift Container Platform cluster. It can not take into account custom requirements. Therefore, you should create a full backup and restore procedure. To prevent data loss, necessary precautions should be taken.

Note that the etcd backup still has all the references to the storage volumes. When you restore etcd, OpenShift Container Platform starts launching the previous pods on nodes and reattaching the same storage. This is really no different than the process of when you remove a node from the cluster and add a new one back in its place. Anything attached to that node will be reattached to the pods on whatever nodes they get rescheduled to.

重要

Backup and restore is not guaranteed. You are responsible for backing up your own data.

35.2. 前提条件

1. Because the restore procedure involves a complete reinstallation, save all the files used in the initial installation. This may include:
   - **~/.config/openshift/installer.cfg.yml** (from the Quick Installation method)
   - Ansible playbooks and inventory files (from the Advanced Installation method)
   - **/etc/yum.repos.d/ose.repo** (from the Disconnected Installation method)
2. Backup the procedures for post-installation steps. Some installations may involve steps that are not included in the installer. This may include changes to the services outside of the control of OpenShift Container Platform or the installation of extra services like monitoring agents. Additional configuration that is not supported yet by the advanced installer might also be affected, for example when using multiple authentication providers.

3. Install packages that provide various utility commands:

```
# yum install etcd
```

4. If using a container-based installation, pull the etcd image instead:

```
# docker pull rhel7/etcd
```

Note the location of the **etcd** data directory (or **$ETCD_DATA_DIR** in the following sections), which depends on how **etcd** is deployed.

Deployment Type	Data Directory
all-in-one cluster	**/var/lib/origin/openshift.local.etcd**
external etcd (located either on a master or another host)	**/var/lib/etcd**

警告

Embedded etcd is no longer supported starting with OpenShift Container Platform 3.7.

35.3. CLUSTER BACKUP

35.3.1. Master Backup

1. Save all the certificates and keys, on each master:

```
# cd /etc/origin/master
# tar cf /tmp/certs-and-keys-$(hostname).tar *.key *.crt
```

35.3.2. Etcd Backup

1. If **etcd** is running on more than one host, stop it on each host:

```
# sudo systemctl stop etcd
```

Although this step is not strictly necessary, doing so ensures that the **etcd** data is fully synchronized.

2. Create an **etcd** backup:

```
# etcdctl backup \
    --data-dir $ETCD_DATA_DIR \
    --backup-dir $ETCD_DATA_DIR.bak
```

注記

If **etcd** is running on more than one host, the various instances regularly synchronize their data, so creating a backup for one of them is sufficient.

注記

For a container-based installation, you must use **docker exec** to run **etcdctl** inside the container.

3. Copy the **db** file over to the backup you created:

```
# cp "$ETCD_DATA_DIR"/member/snap/db "$ETCD_DATA_DIR.bak"/member/snap/db
```

35.3.3. Registry Certificates Backup

1. Save all the registry certificates, on every master and node host.

```
# cd /etc/docker/certs.d/
# tar cf /tmp/docker-registry-certs-$(hostname).tar *
```

注記

When working with one or more external secured registry, any host required to pull or push images must trust registry certificates in order to run pods.

35.4. CLUSTER RESTORE FOR SINGLE-MEMBER ETCD CLUSTERS

To restore the cluster:

1. Reinstall OpenShift Container Platform.
   This should be done in the same way that OpenShift Container Platform was previously installed.

2. Run all necessary post-installation steps.

3. Restore the certificates and keys, on each master:

```
# cd /etc/origin/master
# tar xvf /tmp/certs-and-keys-$(hostname).tar
```

4. Restore from the **etcd** backup:

```
# mv $ETCD_DATA_DIR $ETCD_DATA_DIR.orig
# cp -Rp $ETCD_DATA_DIR.bak $ETCD_DATA_DIR
# chcon -R --reference $ETCD_DATA_DIR.orig $ETCD_DATA_DIR
# chown -R etcd:etcd $ETCD_DATA_DIR
```

5. Create the new single node cluster using etcd's **--force-new-cluster** option. You can do this using the values from **/etc/etcd/etcd.conf**, or you can temporarily modify the **systemd** unit file and start the service normally.
   To do so, edit the **/usr/lib/systemd/system/etcd.service** file, and add **--force-new-cluster**:

```
# sed -i '/ExecStart/s/"$/  --force-new-cluster"/' /usr/lib/systemd/system/etcd.service
# systemctl show etcd.service --property ExecStart --no-pager

ExecStart=/bin/bash -c "GOMAXPROCS=$(nproc) /usr/bin/etcd --force-new-cluster"
```

Then, restart the **etcd** service:

```
# systemctl daemon-reload
# systemctl start etcd
```

6. Verify the **etcd** service started correctly, then re-edit the **/usr/lib/systemd/system/etcd.service** file and remove the **--force-new-cluster** option:

```
# sed -i '/ExecStart/s/ --force-new-cluster//' /usr/lib/systemd/system/etcd.service
# systemctl show etcd.service --property ExecStart --no-pager

ExecStart=/bin/bash -c "GOMAXPROCS=$(nproc) /usr/bin/etcd"
```

7. Restart the **etcd** service, then verify the etcd cluster is running correctly and displays OpenShift Container Platform's configuration:

```
# systemctl daemon-reload
# systemctl restart etcd
```

35.5. CLUSTER RESTORE FOR MULTIPLE-MEMBER ETCD CLUSTERS

If you want to deploy etcd on master hosts, then there is no need to create a new host. If you want to deploy dedicated etcd out of master hosts, then you must create new hosts.

Choose a system to be the initial etcd member, and restore its etcd backup and configuration:

1. Run the following on the etcd host:

```
# ETCD_DIR=/var/lib/etcd/
# mv $ETCD_DIR /var/lib/etcd.orig
# cp -Rp var/lib/etcd.orig/openshift-backup-pre-upgrade/ $ETCD_DIR
# chcon -R --reference /var/lib/etcd.orig/ $ETCD_DIR
# chown -R etcd:etcd $ETCD_DIR
```

2. Restore your **/etc/etcd/etcd.conf** file from backup or **.rpmsave**.

3. Depending on your environment, follow the instructions for Containerized etcd Deployments or Non-Containerized etcd Deployments.

35.5.1. Containerized etcd Deployments

1. Create the new single node cluster using etcd's **--force-new-cluster** option. You can do this with a long, complex command using the values from **/etc/etcd/etcd.conf**, or you can temporarily modify the **systemd** unit file and start the service normally.
To do so, edit the **/etc/systemd/system/etcd_container.service** file, and add **--force-new-cluster**:

```
# sed -i '/ExecStart=/s/$/  --force-new-cluster/' /etc/systemd/system/etcd_container.service
```

```
ExecStart=/usr/bin/docker run --name etcd --rm -v \
/var/lib/etcd:/var/lib/etcd:z -v /etc/etcd:/etc/etcd:ro --env-file=/etc/etcd/etcd.conf \
--net=host --entrypoint=/usr/bin/etcd rhel7/etcd:3.1.9  --force-new-cluster
```

Then, restart the **etcd** service:

```
# systemctl daemon-reload
# systemctl start etcd_container
```

2. Verify the **etcd** service started correctly, then re-edit the **/etc/systemd/system/etcd_container.service** file and remove the **--force-new-cluster** option:

```
# sed  -i '/ExecStart=/s/ --force-new-cluster//' /etc/systemd/system/etcd_container.service

ExecStart=/usr/bin/docker run --name etcd --rm -v /var/lib/etcd:/var/lib/etcd:z -v \
/etc/etcd:/etc/etcd:ro --env-file=/etc/etcd/etcd.conf --net=host \
--entrypoint=/usr/bin/etcd rhel7/etcd:3.1.9
```

3. Restart the **etcd** service, then verify the etcd cluster is running correctly and displays OpenShift Container Platform's configuration:

```
# systemctl daemon-reload
# systemctl restart etcd_container
# etcdctl --cert-file=/etc/etcd/peer.crt \
    --key-file=/etc/etcd/peer.key \
    --ca-file=/etc/etcd/ca.crt \
    --peers="https://172.16.4.18:2379,https://172.16.4.27:2379" \
    ls /
```

4. If you have additional etcd members to add to your cluster, continue to Adding Additional etcd Members. Otherwise, if you only want a single node external etcd, continue to Bringing OpenShift Container Platform Services Back Online.

35.5.2. Non-Containerized etcd Deployments

1. Create the new single node cluster using etcd's **--force-new-cluster** option. You can do this with a long, complex command using the values from **/etc/etcd/etcd.conf**, or you can temporarily modify the **systemd** unit file and start the service normally.
To do so, edit the **/usr/lib/systemd/system/etcd.service** file, and add **--force-new-cluster**:

```
# sed -i '/ExecStart/s/"$/  --force-new-cluster"/' /usr/lib/systemd/system/etcd.service
# systemctl show etcd.service --property ExecStart --no-pager

ExecStart=/bin/bash -c "GOMAXPROCS=$(nproc) /usr/bin/etcd --force-new-cluster"
```

Then restart the **etcd** service:

```
# systemctl daemon-reload
# systemctl start etcd
```

2. Verify the **etcd** service started correctly, then re-edit the **/usr/lib/systemd/system/etcd.service** file and remove the **--force-new-cluster** option:

```
# sed -i '/ExecStart/s/ --force-new-cluster//' /usr/lib/systemd/system/etcd.service
# systemctl show etcd.service --property ExecStart --no-pager

ExecStart=/bin/bash -c "GOMAXPROCS=$(nproc) /usr/bin/etcd"
```

3. Restart the **etcd** service, then verify the etcd cluster is running correctly and displays OpenShift Container Platform's configuration:

```
# systemctl daemon-reload
# systemctl restart etcd
# etcdctl --cert-file=/etc/etcd/peer.crt \
    --key-file=/etc/etcd/peer.key \
    --ca-file=/etc/etcd/ca.crt \
    --peers="https://172.16.4.18:2379,https://172.16.4.27:2379" \
    ls /
```

4. If you have additional etcd members to add to your cluster, continue to Adding Additional etcd Members. Otherwise, if you only want a single node external etcd, continue to Bringing OpenShift Container Platform Services Back Online.

35.5.3. Adding Additional etcd Members

To add additional etcd members to the cluster, you must first adjust the default **localhost** peer in the **peerURLs** value for the first member:

1. **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" \
    member list
```

2. Update the value of **peerURLs** using the **etcdctl member update** command by passing the member ID obtained from the previous step:

```
# 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
```

Alternatively, you can use **curl**:

```
# curl --cacert /etc/etcd/ca.crt \
    --cert /etc/etcd/peer.crt \
    --key /etc/etcd/peer.key \
    https://172.18.1.18:2379/v2/members/511b7fb6cc0001 \
    -XPUT -H "Content-Type: application/json" \
    -d '{"peerURLs":["https://172.18.1.18:2380"]}'
```

3. **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。

4. Now, add each additional member to the cluster one at a time.

警告

Each member must be fully added and brought online one at a time. When adding each additional member to the cluster, the **peerURLs** list must be correct for that point in time, so it will grow by one for each member added. The **etcdctl member add** command will output the values that need to be set in the **etcd.conf** file as you add each member, as described in the following instructions.

a. For each member, add it to the cluster using the values that can be found in that system's **etcd.conf** file:

```
# etcdctl --cert-file=/etc/etcd/peer.crt \
    --key-file=/etc/etcd/peer.key \
    --ca-file=/etc/etcd/ca.crt \
    --peers="https://172.16.4.18:2379,https://172.16.4.27:2379" \
    member add 10.3.9.222 https://172.16.4.27:2380

Added member named 10.3.9.222 with ID 4e1db163a21d7651 to cluster

ETCD_NAME="10.3.9.222"
ETCD_INITIAL_CLUSTER="10.3.9.221=https://172.16.4.18:2380,10.3.9.222=https://172.1
6.4.27:2380"
ETCD_INITIAL_CLUSTER_STATE="existing"
```

b. Using the environment variables provided in the output of the above **etcdctl member add** command, edit the **/etc/etcd/etcd.conf** file on the member system itself and ensure these settings match.

c. Now start etcd on the new member:

```
# rm -rf /var/lib/etcd/member
# systemctl enable etcd
# systemctl start etcd
```

d. Ensure the service starts correctly and the etcd cluster is now healthy:

```
# etcdctl --cert-file=/etc/etcd/peer.crt \
    --key-file=/etc/etcd/peer.key \
    --ca-file=/etc/etcd/ca.crt \
    --peers="https://172.16.4.18:2379,https://172.16.4.27:2379" \
    member list

51251b34b80001: name=10.3.9.221 peerURLs=https://172.16.4.18:2380
clientURLs=https://172.16.4.18:2379
d266df286a41a8a4: name=10.3.9.222 peerURLs=https://172.16.4.27:2380
```

```
clientURLs=https://172.16.4.27:2379

# etcdctl --cert-file=/etc/etcd/peer.crt \
    --key-file=/etc/etcd/peer.key \
    --ca-file=/etc/etcd/ca.crt \
    --peers="https://172.16.4.18:2379,https://172.16.4.27:2379" \
    cluster-health

cluster is healthy
member 51251b34b80001 is healthy
member d266df286a41a8a4 is healthy
```

e. Now repeat this process for the next member to add to the cluster.

5. After all additional etcd members have been added, continue to Bringing OpenShift Container Platform Services Back Online.

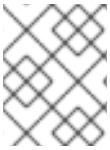
35.6. ADDING NEW ETCD HOSTS

In cases where etcd members have failed and you still have a quorum of etcd cluster members running, you can use the surviving members to add additional etcd members without downtime.

Suggested Cluster Size

Having a cluster with an odd number of etcd hosts can account for fault tolerance. Having an odd number of etcd hosts does not change the number needed for a quorum, but increases the tolerance for failure. For example, a cluster size of three members, quorum is two leaving a failure tolerance of one. This ensures the cluster will continue to operate if two of the members are healthy.

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



注記

The following presumes you have a backup of the **/etc/etcd** configuration for the etcd hosts.

1. If the new etcd members will also be OpenShift Container Platform nodes, see Add the desired number of hosts to the cluster. The rest of this procedure presumes you have added just one host, but if adding multiple, perform all steps on each host.

2. Upgrade etcd and iptables on the surviving nodes:

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

Ensure version **etcd-2.3.7-4.el7.x86_64** or greater is installed, and that the same version is installed on each host.

3. Install etcd and iptables on the new host

```
# yum install etcd iptables-services
```

Ensure version **etcd-2.3.7-4.el7.x86_64** or greater is installed, and that the same version is installed on the new host.

4. Backup the etcd data store on surviving hosts before making any cluster configuration changes.

5. If replacing a failed etcd member, remove the failed member **before** adding the new member.

```
# 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>
```

   Stop the etcd service on the failed etcd member:

```
# systemctl stop etcd
```

6. On the new host, add the appropriate iptables rules:

```
# 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 > /etc/sysconfig/iptables
```

7. Generate the required certificates for the new host. On a surviving etcd host:

   a. Make a backup of the **/etc/etcd/ca/** directory.

   b. Set the variables and working directory for the certificates, ensuring to create the **PREFIX** directory if one has not been created:

```
# cd /etc/etcd
# export NEW_ETCD="<NEW_HOST_NAME>"

# export CN=$NEW_ETCD
# export SAN="IP:<NEW_HOST_IP>"
# export PREFIX="./generated_certs/etcd-$CN/"
```

   c. Create the $PREFIX directory:

```
$ mkdir -p $PREFIX
```

   d. Create the **server.csr** and **server.crt** certificates:

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

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

e. Create the **peer.csr** and **peer.crt** certificates:

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

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

f. Copy the **etcd.conf** and **ca.crt** files, and archive the contents of the directory:

```
# cp etcd.conf ${PREFIX}
# cp ca.crt ${PREFIX}
# tar -czvf ${PREFIX}${CN}.tgz -C ${PREFIX} .
```

g. Transfer the files to the new etcd hosts:

```
# scp ${PREFIX}${CN}.tgz  $CN:/etc/etcd/
```

8. While still on the surviving etcd host, add the new host to the cluster:

a. Add the new host to the cluster:

```
# export ETCD_CA_HOST="<SURVIVING_ETCD_HOSTNAME>"
# export NEW_ETCD="<NEW_ETCD_HOSTNAME>"
# export NEW_ETCD_IP="<NEW_HOST_IP>"

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

ETCD_NAME="<NEW_ETCD_HOSTNAME>"
ETCD_INITIAL_CLUSTER="
<NEW_ETCD_HOSTNAME>=https://<NEW_HOST_IP>:2380,
<SURVIVING_ETCD_HOST>=https:/<SURVIVING_HOST_IP>:2380
ETCD_INITIAL_CLUSTER_STATE="existing"
```

Copy the three environment variables in the etcdctl member add output. They will be used later.

b. On the new host, extract the copied configuration data and set the permissions:

```
# tar -xf /etc/etcd/<NEW_ETCD_HOSTNAME>.tgz -C /etc/etcd/ --overwrite
# chown -R etcd:etcd /etc/etcd/*
```

c. On the new host, remove any etcd data:

```
# rm -rf /var/lib/etcd/member
# chown -R etcd:etcd /var/lib/etcd
```

9. On the new etcd host's **etcd.conf** file:

a. Replace the following with the values generated in the previous step:

- ETCD_NAME
- ETCD_INITIAL_CLUSTER
- ETCD_INITIAL_CLUSTER_STATE
  Replace the IP address with the "NEW_ETCD" value for:
- ETCD_LISTEN_PEER_URLS
- ETCD_LISTEN_CLIENT_URLS
- ETCD_INITIAL_ADVERTISE_PEER_URLS
- ETCD_ADVERTISE_CLIENT_URLS
  For replacing failed members, you will need to remove the failed hosts from the etcd configuration.

10. 新規ホストで etcd を起動します。

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

11. To verify that the new member has been added successfully:

```
etcdctl -C https://${ETCD_CA_HOST}:2379 --ca-file=/etc/etcd/ca.crt \
  --cert-file=/etc/etcd/peer.crt     \
  --key-file=/etc/etcd/peer.key cluster-health
```

12. Update the master configuration on all masters to point to the new etcd host

a. On every master in the cluster, edit **/etc/origin/master/master-config.yaml**

b. Find the **etcdClientInfo** section.

c. Add the new etcd host to the **urls** list.

d. If a failed etcd host was replaced, remove it from the list.

e. Restart the master API service.
On each master:

```
# systemctl restart atomic-openshift-master-api atomic-openshift-master-controllers
```

The procedure to add an etcd member is complete.

35.7. BRINGING OPENSHIFT CONTAINER PLATFORM SERVICES BACK ONLINE

On each OpenShift Container Platform master, restore your master and node configuration from backup and enable and restart all relevant services.

```
# cp /etc/sysconfig/atomic-openshift-master-api.rpmsave /etc/sysconfig/atomic-openshift-master-api
# cp /etc/sysconfig/atomic-openshift-master-controllers.rpmsave /etc/sysconfig/atomic-openshift-
master-controllers
# cp /etc/origin/master/master-config.yaml.<timestamp> /etc/origin/master/master-config.yaml
# cp /etc/origin/node/node-config.yaml.<timestamp> /etc/origin/node/node-config.yaml
# cp /etc/origin/master/scheduler.json.<timestamp> /etc/origin/master/scheduler.json
# systemctl enable atomic-openshift-master-api
# systemctl enable atomic-openshift-master-controllers
# systemctl enable atomic-openshift-node
# systemctl start atomic-openshift-master-api
# systemctl start atomic-openshift-master-controllers
# systemctl start atomic-openshift-node
```

On each OpenShift Container Platform node, restore your **node-config.yaml** file from backup and enable and restart the **atomic-openshift-node** service:

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

Your OpenShift Container Platform cluster should now be back online.

35.8. PROJECT BACKUP

A future release of OpenShift Container Platform will feature specific support for per-project back up and restore.

For now, to back up API objects at the project level, use **oc export** for each object to be saved. For example, to save the deployment configuration **frontend** in YAML format:

```
$ oc export dc frontend -o yaml > dc-frontend.yaml
```

To back up all of the project (with the exception of cluster objects like namespaces and projects):

```
$ oc export all -o yaml > project.yaml
```

35.8.1. Role Bindings

Sometimes custom policy role bindings are used in a project. For example, a project administrator can give another user a certain role in the project and grant that user project access.

These role bindings can be exported:

```
$ oc get rolebindings -o yaml --export=true > rolebindings.yaml
```

35.8.2. Service Accounts

If custom service accounts are created in a project, these need to be exported:

```
$ oc get serviceaccount -o yaml --export=true > serviceaccount.yaml
```

35.8.3. Secrets

Custom secrets like source control management secrets (SSH Public Keys, Username/Password) should be exported if they are used:

```
$ oc get secret -o yaml --export=true > secret.yaml
```

35.8.4. Persistent Volume Claims

If the application within a project uses a persistent volume through a persistent volume claim (PVC), these should be backed up:

```
$ oc get pvc -o yaml --export=true > pvc.yaml
```

35.9. PROJECT RESTORE

To restore a project, recreate the project and recreate all of the objects that were exported during the backup:

```
$ oc new-project myproject
$ oc create -f project.yaml
$ oc create -f secret.yaml
$ oc create -f serviceaccount.yaml
$ oc create -f pvc.yaml
$ oc create -f rolebindings.yaml
```

注記

Some resources can fail to be created (for example, pods and default service accounts).

35.10. APPLICATION DATA BACKUP

In many cases, application data can be backed up using the **oc rsync** command, assuming **rsync** is installed within the container image. The Red Hat **rhel7** base image does contain **rsync**. Therefore, all images that are based on **rhel7** contain it as well. See Troubleshooting and Debugging CLI Operations - rsync.

警告

This is a **generic** backup of application data and does not take into account application-specific backup procedures, for example, special export/import procedures for database systems.

Other means of backup may exist depending on the type of the persistent volume (for example, Cinder, NFS, Gluster, or others).

The paths to back up are also **application specific**. You can determine what path to back up by looking at the **mountPath** for volumes in the **deploymentconfig**.

Example of Backing up a Jenkins Deployment's Application Data

1. Get the application data **mountPath** from the **deploymentconfig**:

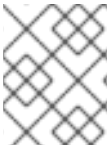
```
$ oc get dc/jenkins -o jsonpath='{ .spec.template.spec.containers[?
(@.name=="jenkins")].volumeMounts[?(@.name=="jenkins-data")].mountPath }'
/var/lib/jenkins
```

2. Get the name of the pod that is currently running:

```
$ oc get pod --selector=deploymentconfig=jenkins -o jsonpath='{ .metadata.name }'
jenkins-1-37nux
```

3. Use the **oc rsync** command to copy application data:

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



注記

This type of application data backup can only be performed while an application pod is currently running.

35.11. APPLICATION DATA RESTORE

The process for restoring application data is similar to the application backup procedure using the **oc rsync** tool. The same restrictions apply and the process of restoring application data requires a persistent volume.

Example of Restoring a Jenkins Deployment's Application Data

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. Restart the application with new data (**optional**):

```
$ 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します。

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

第36章 OPENSHIFT SDN のトラブルシューティング

36.1. 概要

As described in the SDN documentation there are multiple layers of interfaces that are created to correctly pass the traffic from one container to another. In order to debug connectivity issues, you have to test the different layers of the stack to work out where the problem arises. This guide will help you dig down through the layers to identify the problem and how to fix it.

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。

36.2. 用語

クラスター

クラスター内の一連のマシンです。**例:** マスターおよびノード。

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ー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。

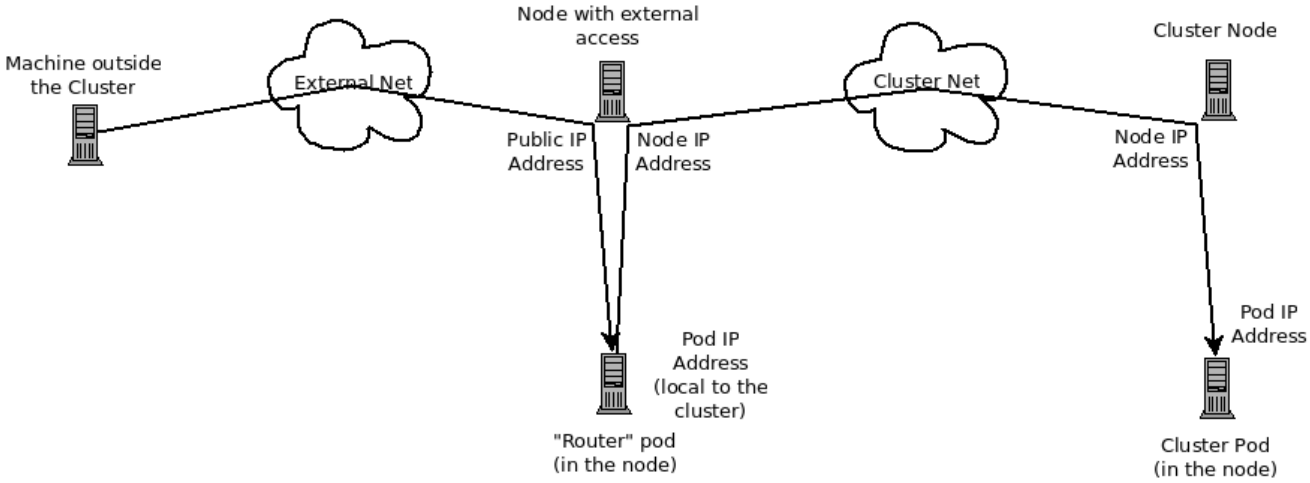
Pod アドレス

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構成部分を示しています。



36.3. HTTP サービスへの外部アクセスのデバッグ

If you are on an machine outside the cluster and are trying to access a resource provided by the cluster there needs to be a process running in a pod that listens on a public IP address and "routes" that traffic inside the cluster. The OpenShift Container Platform router serves that purpose for HTTP, HTTPS (with SNI), WebSockets, or TLS (with SNI).

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。**ctrl-]** を押してから **enter** キーを押し、**close** を入力して telnet を終了します。「ルーターのデバッグ」 セクションに移行してルーターの他のものを確認します。

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。

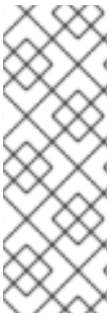
または、以下が表示される場合があります。

```
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。

36.4. 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

If you have access to administrator credentials but are no longer logged in as the default system user **system:admin**, you can log back in as this user at any time as long as the credentials are still present in your CLI configuration file. The following command logs in and switches to the **default** project:

```
$ 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}}{{end}}{{"\n"}}'
HostIP: 192.168.122.202    PodIP: 10.128.0.4
```

You should see the host IP that corresponds to your external address. If you do not, refer to the router documentation to configure the router pod to run on the right node (by setting the affinity correctly) or update your DNS to match the IP addresses where the routers are running.

(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
```

If the host name and path from your URL don't match anything in the list of returned routes, then you need to add a route. See the router documentation.

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。

36.5. 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
```

You should see your service in the list. If not, then you need to define your service.

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、**それぞれの** エンドポイントについて以下を試行します。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」 セクションに移行してください。

すべてが機能する場合は、「Kubernetes のデバッグ」 セクションに移行してサービス IP アドレスが機能しない理由を判別します。

36.6. 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. 事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** エントリーを検索し、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ー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ースの詳細を取得し、このインターフェースが予想されるインターフェー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。

- どちらの側にもイーサネットリンクがあるか? **ethtool <network_interface>** 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どちらの側でも適切に設定されているか? **ethtool <network_interface>** 情報の残りの部分を確認します。

- ケーブルは適切にプラグインされているか? 正しいポートに接続されているか?

- スイッチは適切に設定されているか?

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。

36.7. ローカルネットワークのデバッグ

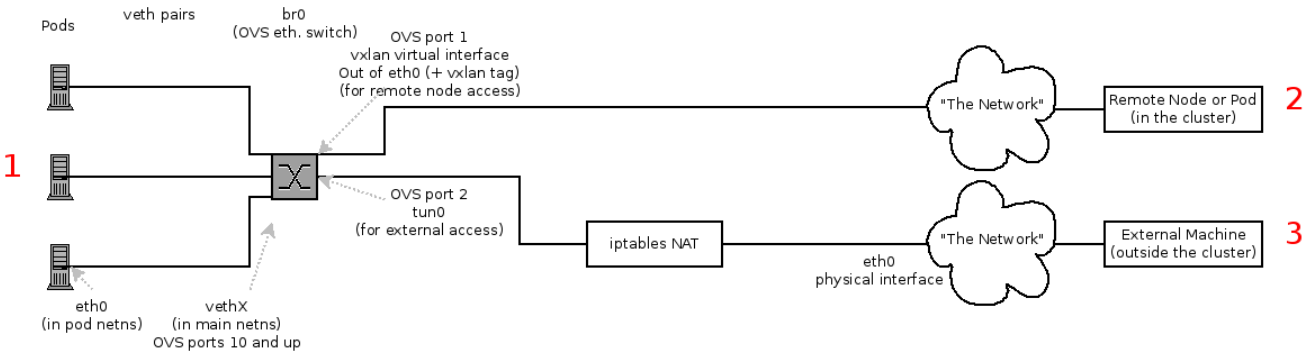
ここで通信できないものの、ノード間通信が設定された 1 つ以上のエンドポイントの一覧が表示されます。それぞれのエンドポイントについて問題点を特定する必要がありますが、まずは SDN が複数の異なる Pod についてノードでネットワークをどのように設定しているかについて理解する必要があります。

36.7.1. ノードのインターフェース

以下は OpenShift SDN が作成するインターフェースです。

- **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に固有ではないフロールールのセットも設定します。
- **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。

36.7.2. 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。

36.7.3. デバッグ手順

36.7.3.1. 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。

36.7.3.2. 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)。

36.7.4. Is the Open vSwitch configured correctly?

Check the Open vSwitch bridges on both sides:

```
# ovs-vsctl list-br
br0
```

This should be **br0**.

You can list all of the ports that ovs knows about:

```
# 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。

```
# 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。

36.7.4.1. 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。

36.7.4.2. 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)。

36.8. 仮想ネットワークのデバッグ

36.8.1. 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ース (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。

```
# cat /etc/origin/node/node-config.yaml
...
```

```
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ースの 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。

36.9. 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
```

That should return the IP address for the github server, but check that you got back the correct address. If you get back no address, or the address of one of your machines, then you may be matching the wildcard entry in yoir local DNS server.

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。

36.10. 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。

36.11. 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。

36.12. 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>
```

36.13. その他の注意点

36.13.1. 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。

36.13.2. 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ース (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。

36.13.3. 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ースは **ethtool -S ifname** で判別できます。
- ドライバータイプ: **ethtool -i ifname**

第37章 診断ツール

37.1. 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。

37.2. 診断ツールの使用

OpenShift Container Platform can be deployed in many ways: built from source, included in a VM image, in a container image, or as enterprise RPMs. Each method implies a different configuration and environment. To minimize environment assumptions, the diagnostics were added to the **openshift** binary so that wherever there is an OpenShift Container Platform server or client, the diagnostics can run in the exact same environment.

診断ツールを使用するには (マスターホスト上で使用するのが望ましい)、クラスター管理者として以下を実行します。

```
$ oc adm diagnostics
```

This runs all available diagnostics, skipping any that do not apply.

You can run one or multiple specific diagnostics by name, or run specific diagnostics by name as you work to address issues. For example:

```
$ oc adm diagnostics <name1> <name2>
```

The options mostly require working configuration files. For example, the **NodeConfigCheck** does not run unless a node configuration is available.

Diagnostics look for configuration files in standard locations:

- クライアント:
  - As indicated by the **$KUBECONFIG** environment variable variable
  - **~/.kube/config file**

- マスター:
  - **/etc/origin/master/master-config.yaml**
- ノード:
  - **/etc/origin/node/node-config.yaml**

Non-standard locations can be specified with flags (respectively, **--config**, **--master-config**, and **--node-config**). If a configuration file is not found or specified, related diagnostics are skipped.

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	Check that the cluster has a working Docker registry for builds and image streams.
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。
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。

診断名	目的
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	Create diagnostic pods on multiple nodes to diagnose common network issues from an application standpoint. For example, this checks that pods can connect to services, other pods, and the external network. 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。

37.3. サーバー環境における診断の実行

Master and node diagnostics are most useful in an Ansible-deployed cluster. This provides some diagnostic benefits:

- 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。

- すべてのコンポーネントが journald に対してログを記録する。

Having configuration files where Ansible places them means that you will generally not need to specify where to find them. Running **oc adm diagnostics** without flags will look for master and node configurations in the standard locations and use them if found; this should make the Ansible-installed use case as simple as possible. Also, it is easy to specify configuration files that are not in the expected locations:

```
$ oc adm diagnostics --master-config=<file_path> --node-config=<file_path>
```

Systemd units and logs entries in journald are necessary for the current log diagnostic logic. For other deployment types, logs may be going into files, to stdout, or may combine node and master. At this time, for these situations, log diagnostics are not able to work properly and will be skipped.

37.4. クライアント環境での診断の実行

You may have access as an ordinary user, and/or as a **cluster-admin** user, and/or may be running on a host where OpenShift Container Platform master or node servers are operating. The diagnostics attempt to use as much access as the user has available.

A client with ordinary access should be able to diagnose its connection to the master and run a diagnostic pod. If multiple users or masters are configured, connections will be tested for all, but the diagnostic pod only runs against the current user, server, or project.

A client with **cluster-admin** access available (for any user, but only the current master) should be able to diagnose the status of infrastructure such as nodes, registry, and router. In each case, running **oc adm diagnostics** looks for the client configuration in its standard location and uses it if available.

37.5. ANSIBLE ベースのヘルスチェック

Additional diagnostic health checks are available through the Ansible-based tooling used to install and manage OpenShift Container Platform clusters. They can report common deployment problems for the current OpenShift Container Platform installation.

These checks can be run either using the **ansible-playbook** command (the same method used during Advanced Installation) or as a containerized version of **openshift-ansible**. For the **ansible-playbook** method, the checks are provided by the **atomic-openshift-utils** RPM package. For the containerized method, the **openshift3/ose-ansible** container image is distributed via the Red Hat Container Registry . Example usage for each method are provided in subsequent sections.

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。

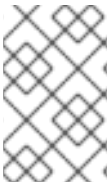
警告

Due to potential changes the health check playbooks could make to hosts, they should only be used on clusters that have been deployed using Ansible and using the same inventory file with which it was deployed. Changes mostly involve installing dependencies so that the checks can gather required information, but it is possible for certain system components (for example, **docker** or networking) to be altered if their current state differs from the configuration in the inventory file. Only run these health checks if you would not expect your inventory file to make any changes to your current cluster configuration.

表37.1 正常性診断チェック

チェック名	目的
etcd_imagedata_size	This check measures the total size of OpenShift Container Platform image data in an etcd cluster. The check fails if the calculated size exceeds a user-defined limit. If no limit is specified, this check will fail if the size of image data amounts to 50% or more of the currently used space in the etcd cluster. A failure from this check indicates that a significant amount of space in etcd is being taken up by OpenShift Container Platform image data, which can eventually result in your etcd cluster crashing. A user-defined limit may be set by passing the **etcd_max_image_data_size_bytes** variable. For example, setting **etcd_max_image_data_size_bytes=40000000000** will cause the check to fail if the total size of image data stored in etcd exceeds 40 GB.
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。 For further information on improving etcd performance, see Recommended Practices for OpenShift Container Platform etcd Hosts and the Red Hat Knowledgebase.

チェック名	目的
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。 The threshold limit for total percent usage can be set with a variable in your inventory file, for example **max_thinpool_data_usage_percent=90**. This also checks that **docker**'s storage is using a supported configuration.
curator、elasticsearch、fluentd、kibana	This set of checks verifies that Curator, Kibana, Elasticsearch, and Fluentd pods have been deployed and are in a **running** state, and that a connection can be established between the control host and the exposed Kibana URL. These checks will only run if the **openshift_logging_install_logging** inventory variable is set to **true**, to ensure that they are executed in a deployment where cluster logging has been enabled.
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。 A user-defined timeout may be set by passing the **openshift_check_logging_index_timeout_seconds** variable. For example, setting **openshift_check_logging_index_timeout_seconds=45** will cause the check to fail if a newly-created log entry is not able to be queried via Elasticsearch after 45 seconds.



注記

A similar set of checks meant to run as part of the installation process can be found in Configuring Cluster Pre-install Checks. Another set of checks for checking certificate expiration can be found in Redeploying Certificates.

37.5.1. ansible-playbook によるヘルスチェックの実行

To run the **openshift-ansible** health checks using the **ansible-playbook** command, specify your cluster's inventory file and run the **health.yml** playbook:

```
# ansible-playbook -i <inventory_file> \
    /usr/share/ansible/openshift-ansible/playbooks/openshift-checks/health.yml
```

コマンドラインに変数を設定するには、**key=value** 形式の必要な変数に **-e** フラグを組み込みます。以下は例になります。

```
# ansible-playbook -i <inventory_file> \
    /usr/share/ansible/openshift-ansible/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カ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。以下は例になります。

```
openshift_disable_check=etcd_traffic,etcd_volume
```

Alternatively, set any checks you want to disable as variables with **-e openshift_disable_check=<check1>,<check2>** when running the **ansible-playbook** command.

37.5.2. Docker CLI でのヘルスチェックの実行

It is possible to run the **openshift-ansible** playbooks in a Docker container, avoiding the need for installing and configuring Ansible, on any host that can run the **ose-ansible** image via the Docker CLI.

To do so, specify your cluster's inventory file and the **health.yml** playbook when running the following **docker run** command as a non-root user that has privileges to run containers:

```
# 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. Change **/etc/ansible/hosts** to the location of your cluster's inventory file, if different. This file will be bind-mounted to **/tmp/inventory**, which is used according to the **INVENTORY_FILE**

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。

In the above command, the SSH key is mounted with the **:Z** flag so that the container can read the SSH key from its restricted SELinux context; this means that your original SSH key file will be relabeled to something like **system_u:object_r:container_file_t:s0:c113,c247**. For more details about **:Z**, see the **docker-run(1)** man page.

Keep this in mind for these volume mount specifications because it could have unexpected consequences. For example, if you mount (and therefore relabel) your **$HOME/.ssh** directory, **sshd** will become unable to access your public keys to allow remote login. To avoid altering the original file labels, mounting a copy of the SSH key (or directory) is recommended.

You might want to mount an entire **.ssh** directory for various reasons. For example, this would allow you to use an SSH configuration to match keys with hosts or modify other connection parameters. It would also allow you to provide a **known_hosts** file and have SSH validate host keys, which is disabled by the default configuration and can be re-enabled with an environment variable by adding **-e ANSIBLE_HOST_KEY_CHECKING=True** to the **docker** command line.

第38章 アプリケーションのアイドリング

38.1. 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構成されています。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。

38.2. 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 コマンドを実行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。

38.2.1. 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>
```

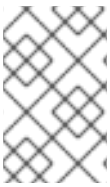
38.2.2. 複数サービスのアイドリング

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。

38.3. 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39章 クラスター容量の分析

39.1. 概要

As a cluster administrator, you can use the cluster capacity tool to view the number of pods that can be scheduled to increase the current resources before they become exhausted, and to ensure any future pods can be scheduled. This capacity comes from an individual node host in a cluster, and includes CPU, memory, disk space, and others.

The cluster capacity tool simulates a sequence of scheduling decisions to determine how many instances of an input pod can be scheduled on the cluster before it is exhausted of resources to provide a more accurate estimation.

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。

You can run the cluster capacity analysis tool as a stand-alone utility from the command line, or as a job in a pod inside an OpenShift Container Platform cluster. Running it as job inside of a pod enables you to run it multiple times without intervention.

39.2. コマンドラインでのクラスター容量分析の実行

To run the tool on the command line:

```
$ cluster-capacity --kubeconfig <path-to-kubeconfig> \
    --podspec <path-to-pod-spec>
```

The **--kubeconfig** option indicates your Kubernetes configuration file, and the **--podspec** option indicates a sample pod specification file, which the tool uses for estimating resource usage. The **podspec** specifies its resource requirements as **limits** or **requests**. The cluster capacity tool takes the pod's resource requirements into account for its estimation analysis.

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。

```
$ 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。

39.3. 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
```

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=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: openshift/origin-cluster-capacity
          imagePullPolicy: "Always"
```

```
      volumeMounts:
      - mountPath: /test-pod
        name: test-volume
      env:
      - name: CC_INCLUSTER 1
        value: "true"
      command:
      - "/bin/sh"
      - "-ec"
      - |
        /bin/cluster-capacity --podspec=/test-pod/pod.yaml --verbose
    restartPolicy: "Never"
    serviceAccountName: cluster-capacity-sa
    volumes:
    - name: test-volume
      configMap:
        name: cluster-capacity-configmap
```

1 クラスター容量ツールにクラスター内で Pod として実行されていることを認識させる環境変数です。
ConfigMap の **pod.yaml** キーは Pod 仕様ファイル名と同じですが、これは必須ではありません。これを実行することで、入力 Pod 仕様ファイルは **/test-pod/pod.yaml** として Pod 内でアクセスできます。

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 の数を確認します。

```
$ 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)
```