OpenShift Container Platform 3.10

Day 2 操作ガイド

OpenShift Container Platform 3.10 Day 2 操作ガイド

Last Updated: 2018-11-14

OpenShift Container Platform 3.10 Day 2 操作ガイド

OpenShift Container Platform 3.10 Day 2 操作ガイド

法律上の通知

概要

『OpenShift Container Platform クラスター管理』ガイドは設定に重点を当てていますが、本書では日常的に実行される一般的なメンテナンスタスクの概要について説明します。

目次

第1章 概要

このセクションは、新規インストールを扱う OpenShift Container Platform の管理者向けに用意されています。

『OpenShift Container Platform クラスター』ガイドは設定に重点を当てていますが、本書では日常的に実行される一般的なメンテナンスタスクについて概説します。

第2章 1 回実行 (RUN-ONCE) タスク

OpenShift Container Platform のインストール後、ホストのスムーズな実行を維持するためにシステムへの追加の設定が必要になる場合があります。

これらは 1 回実行 (run-once) タスクとして分類され、これらのタスクは状況の変更に応じていつでも実行できます。

2.1. NTP 同期

NTP (ネットワークタイムプロトコル) は、常にホストを世界時計と同期します。時間の同期は、ログの記録やタイムスタンプなどの時間に依存する操作に重要であり、OpenShift Containter Platform のビルドに使用される Kubernetes で使用することが強く推奨されます。OpenShift Container Platform の操作には etcd リーダーの選択、Pod およびその他の問題のヘルスチェックが含まれ、これらは時間のずれの発生を防ぐのに役立ちます。

インスタンスによっては、NTP がデフォルトで有効にされていない場合があります。ホストが NTP を使用するように設定されていることを確認するには、以下を実行します。

```
$ timedatectl
      Local time: Thu 2017-12-21 14:58:34 UTC
  Universal time: Thu 2017-12-21 14:58:34 UTC
        RTC time: Thu 2017-12-21 14:58:34
       Time zone: Etc/UTC (UTC, +0000)
     NTP enabled: yes
NTP synchronized: yes
 RTC in local TZ: no
      DST active: n/a
```

`NTP enabled` と **`NTP synchronized`** の両方が **`yes`** の場合、NTP 同期は有効にされています。

`no` の場合、**`ntp`** または **`chrony`** RPM パッケージをインストールし、有効にします。

NTP の場合:

```
# timedatectl set-ntp true
```

chrony の場合:

```
# yum install chrony
# systemctl enable chronyd --now
```

重要

時間の同期は、NTP を使用しているか、その他の方法を使用しているかにかかわらず、クラスター内のすべてのホストで有効にされている必要があります。

`timedatectl` コマンド、タイムゾーンおよび時計の同期についての詳細は、「日付と時刻の設定」および「UTC、タイムゾーン、および DST」を参照してください。

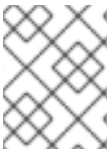
2.2. エントロピー

OpenShift Container Platform はエントロピーを使用して ID または SSL トラフィックなどのオブジェクトの乱数を生成します。これらの操作はタスクを完了するのに十分なエントロピーが用意されるまで待機します。十分なエントロピーがないと、カーネルは適切なスピードでこれらの乱数を生成することができません。これにより、タイムアウトが生じたり、セキュアな接続が拒否される可能性があります。

利用可能なエントロピーを確認するには、以下を実行します。

```
$ cat /proc/sys/kernel/random/entropy_avail
2683
```

利用可能なエントロピーはクラスター内のすべてのホストで検証する必要があります。理想的には、この値は **1000** より大きい値に指定してください。



注記

Red Hat では、この値をモニターすること、およびこの値が **800** 未満の場合には警告を発行することを推奨しています。

または、**rngtest** コマンドを使用すると、十分なエントロピーだけでなく、システムが十分なエントロピーを **フィード** できるかどうかを確認できます。

```
$ cat /dev/random | rngtest -c 100
```

rngtest コマンドは **rng-tools** で利用できます。

上記のタスクの完了に約 30 秒の時間がかかる場合、利用可能なエントロピーが十分にないことを示しています。

ご使用の環境によっては、複数の方法でエントロピーを増やすことができます。詳細については、こちらのブログ (https://developers.redhat.com/blog/2017/10/05/entropy-rhel-based-cloud-instances/) を参照してください。

通常は **rng-tools** パッケージをインストールし、**rngd** サービスを有効にしてエントロピーを増大させることができます。

```
# yum install rng-tools
# systemctl enable --now rngd
```

rngd サービスが起動すると、エントロピーは十分なレベルに引き上げられるはずです。

2.3. デフォルトストレージクラスのチェック

動的にプロビジョニングされる永続ストレージの適切な機能を維持するには、デフォルトのストレージグラスを定義しておく必要があります。インストール時に、このデフォルトストレージクラスは Amazon Web Services (AWS)、Google Cloud Platform (GCP) などの共通のクラウドプロバイダーについて定義されます。

デフォルトストレージクラスが定義されていることを確認するには、以下を実行します。

```
$ oc get storageclass
NAME                 TYPE
ssd                  kubernetes.io/gce-pd
```

```
standard (default)   kubernetes.io/gce-pd
```

上記は GCP で実行されている OpenShift Container Platform インスタンスからの出力を抜粋したものです。この OpenShift Container Platform インスタンスでは、標準 (HDD) および SSD の 2 種類の永続ストレージが利用可能です。標準ストレージクラスはデフォルトとして設定されることに注意してください。ストレージクラスが定義されていない場合や、デフォルトとして何も設定されていない場合には、「動的プロビジョニングとストレージクラスの作成」のセクションを参照し、ストレージクラスの設定方法を確認してください。

第3章 環境ヘルスチェック

このトピックでは、OpenShift Container Platform クラスターおよび各種コンポーネントの全体的な健全性を確認する手順について、また予想される動作について説明します。

各種コンポーネントの検証プロセスについて把握することは、問題のトラブルシューティングにおける最初のステップになります。問題が発生している場合には、このセクションで提供されるチェックを使用して問題を診断できます。

3.1. 全体的な環境ヘルスチェック

OpenShift Container Platform クラスターの全体的な機能を確認するために、アプリケーションのサンプルをビルドし、デプロイします。

手順

1. **validate** という名前の新規プロジェクト、および **cakephp-mysql-example** テンプレートからアプリケーションのサンプルを作成します。

```
$ oc new-project validate
$ oc new-app cakephp-mysql-example
```

ログを確認してからビルドに進みます。

```
$ oc logs -f bc/cakephp-mysql-example
```

2. ビルドが完了すると、データベースとアプリケーションの 2 つの Pod が実行されるはずです。

```
$ oc get pods
NAME                              READY     STATUS      RESTARTS
AGE
cakephp-mysql-example-1-build     0/1       Completed   0          1m
cakephp-mysql-example-2-247xm     1/1       Running     0
39s
mysql-1-hbk46                     1/1       Running     0
1m
```

3. アプリケーション URL にアクセスします。Cake PHP フレームワークの welcome ページが表示されるはずです。URL では **cakephp-mysql-example-validate.<app_domain>** という形式を使用しています。

4. 機能の確認後は、**validate** プロジェクトを削除できます。

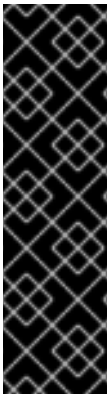
```
$ oc delete project validate
```

プロジェクト内のすべてのリソースも削除されます。

3.2. PROMETHEUS を使用した警告の作成

OpenShift Container Platform と Prometheus を統合して、ビジュアルや警告を作成し、環境の問題が発生する前に診断できるようにします。このような問題として考えられるのは、ノードのダウン、Pod による CPU またはメモリーの過剰使用などです。

詳細は、『クラスター設定ガイド』の「OpenShift Container Platform での Prometheus の設定」セクションを参照してください。



重要

OpenShift Container Platform 上での Prometheus 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。

3.3. ホストの健全性

クラスターが稼働していることを確認するには、マスターインスタンスに接続し、以下を実行します。

```
$ oc get nodes
NAME                 STATUS                     AGE       VERSION
ocp-infra-node-1clj  Ready                      1h
v1.6.1+5115d708d7
ocp-infra-node-86qr  Ready                      1h
v1.6.1+5115d708d7
ocp-infra-node-g8qw  Ready                      1h
v1.6.1+5115d708d7
ocp-master-94zd      Ready,SchedulingDisabled   1h
v1.6.1+5115d708d7
ocp-master-gjkm      Ready,SchedulingDisabled   1h
v1.6.1+5115d708d7
ocp-master-wc8w      Ready,SchedulingDisabled   1h
v1.6.1+5115d708d7
ocp-node-c5dg        Ready                      1h
v1.6.1+5115d708d7
ocp-node-ghxn        Ready                      1h
v1.6.1+5115d708d7
ocp-node-w135        Ready                      1h
v1.6.1+5115d708d7
```

上記のクラスターサンプルから、3 つのマスターホスト、3 つのインフラストラクチャーノードホスト、および 3 つのノードホストで構成されていること、すべて実行中であることが分かります。クラスター内のホストはすべて、この出力に表示されているはずです。

Ready ステータスは、マスターホストがノードホストと通信でき、ノードが Pod を実行できる状態にあることを示します (スケジューリングが無効にされているノードを除く)。

基本的な etcd の健全性のステータスは、任意のマスターインスタンスから **etcdctl2** コマンドを実行して確認できます。

```
# etcdctl2 cluster-health
member 59df5107484b84df is healthy: got healthy result from
https://10.156.0.5:2379
member 6df7221a03f65299 is healthy: got healthy result from
https://10.156.0.6:2379
```

```
member fea6dfedf3eecfa3 is healthy: got healthy result from
https://10.156.0.9:2379
cluster is healthy
```

ただし、関連付けられたマスターホストを含め、etcd ホストについての詳細情報を取得するには、以下を実行します。

```
# etcdctl2 member list
295750b7103123e0: name=ocp-master-zh8d peerURLs=https://10.156.0.7:2380
clientURLs=https://10.156.0.7:2379 isLeader=true
b097a72f2610aea5: name=ocp-master-qcg3 peerURLs=https://10.156.0.11:2380
clientURLs=https://10.156.0.11:2379 isLeader=false
fea6dfedf3eecfa3: name=ocp-master-j338 peerURLs=https://10.156.0.9:2380
clientURLs=https://10.156.0.9:2379 isLeader=false
```

etcd クラスターがマスターサービスと同じ場所に配置されている場合はすべての etcd ホストにマスターホスト名が含まれますが、etcd サービスが別々のホストで実行されている場合はそれらの etcd ホスト名がすべて一覧表示されます。

注記

etcdctl2 は、v2 データモデルの etcd クラスターのクエリーに使用するフラグが含まれる **etcdctl** ツールのエイリアスです (v3 データモデルの場合は **etcdctl3**)。

3.4. ルーターおよびレジストリーの健全性

ルーターサービスが実行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 -n default get deploymentconfigs/router
NAME      REVISION   DESIRED   CURRENT   TRIGGERED BY
router    1          3         3         config
```

DESIRED および **CURRENT** 列の値はノードホストの数に一致しているはずです。

レジストリーのステータスを確認する場合も同じコマンドを使用します。

```
$ oc -n default get deploymentconfigs/docker-registry
NAME              REVISION   DESIRED   CURRENT   TRIGGERED BY
docker-registry   1          3         3         config
```

注記

コンテナーレジストリーのインスタンスを複数実行するには、複数プロセスによる書き込みをサポートするバックエンドストレージが必要です。選択したインフラストラクチャープロバイダーにこの機能が含まれていない場合には、コンテナーレジストリーのインスタンスを 1 つ実行できます。

すべての Pod が実行中であること、およびどのホストで実行中であるかを確認するには、以下を実行します。

```
$ oc -n default get pods -o wide
NAME                       READY     STATUS    RESTARTS   AGE       IP
NODE
```

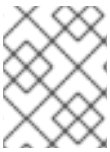
```
docker-registry-1-54nhl    1/1       Running   0          2d
172.16.2.3    ocp-infra-node-tl47
docker-registry-1-jsm2t    1/1       Running   0          2d
172.16.8.2    ocp-infra-node-62rc
docker-registry-1-qbt4g    1/1       Running   0          2d
172.16.14.3   ocp-infra-node-xrtz
registry-console-2-gbhcz   1/1       Running   0          2d
172.16.8.4    ocp-infra-node-62rc
router-1-6zhf8             1/1       Running   0          2d
10.156.0.4    ocp-infra-node-62rc
router-1-ffq4g             1/1       Running   0          2d
10.156.0.10   ocp-infra-node-tl47
router-1-zqxbl             1/1       Running   0          2d
10.156.0.8    ocp-infra-node-xrtz
```

注記

OpenShift Container Platform が外部コンテナーレジストリーを使用している場合、内部レジストリーサービスは実行中である必要がありません。

3.5. ネットワーク接続

ネットワーク接続には、ノードの対話用のクラスターネットワークと Pod の対話用の SDN (Software Defined Network) という 2 つの主要なネットワーク層が含まれます。OpenShift Container Platform は複数のネットワーク設定をサポートし、これらの設定は特定のインフラストラクチャープロバイダー向けに最適化されることがよくあります。



注記

ネットワークが複雑であることから、本書ではすべての検証シナリオについては扱いません。

3.5.1. マスターホストでの接続性

etcd およびマスターホスト

マスターサービスは etcd キー値ストアを使用してそれらの同期状態を維持します。マスターと etcd サービス間の通信は、それらの etcd サービスがマスターホストの同じ場所に置かれている場合でも、etcd サービス用にのみ指定されるホストで実行されている場合でも重要になります。この通信は TCP ポート **2379** および **2380** で実行されます。この通信をチェックする方法については、「ホストの健全性」のセクションを参照してください。

SkyDNS

SkyDNS は、OpenShift Container Platform で実行されるローカルサービスの名前解決を行います。このサービスは **TCP** および **UDP** ポート **8053** を使用します。

名前解決を確認するには、以下を実行します。

```
$ dig +short docker-registry.default.svc.cluster.local
172.30.150.7
```

応答が以下の出力に一致する場合、**SkyDNS** サービスは適切に機能していることになります。

-

```
$ oc get svc/docker-registry -n default
NAME              CLUSTER-IP     EXTERNAL-IP   PORT(S)    AGE
docker-registry   172.30.150.7   <none>        5000/TCP   3d
```

API サービスおよび Web コンソール

API サービスおよび Web コンソールはどちらも同じポート (セットアップによって異なりますが、通常は **TCP 8443** または **443**) を共有します。このポートはクラスター内で、またデプロイされた環境で作業する必要のあるすべてのユーザーにとって利用可能な状態である必要があります。このポートに到達するために使用される URL は内部クラスターおよび外部クライアント用に異なる場合があります。

以下の例では、**https://internal-master.example.com:443** URL は内部クラスターによって使用され、**https://master.example.com:443** URL は外部クライアントによって使用されています。任意のノードホストで以下を実行します。

```
$ curl https://internal-master.example.com:443/version
{
  "major": "1",
  "minor": "6",
  "gitVersion": "v1.6.1+5115d708d7",
  "gitCommit": "fff65cf",
  "gitTreeState": "clean",
  "buildDate": "2017-10-11T22:44:25Z",
  "goVersion": "go1.7.6",
  "compiler": "gc",
  "platform": "linux/amd64"
}
```

これはクライアントのネットワークから到達可能である必要があります。

```
$ curl -k https://master.example.com:443/healthz
ok
```

3.5.2. ノードインスタンスでの接続性

Pod の通信に使用されるノードでの SDN 接続は、デフォルトで **UDP** ポート **4789** を使用します。

ノードホストの機能を確認するには、新規アプリケーションを作成します。以下の例では、ノードがインフラストラクチャーノードで実行されている docker レジストリーに到達できるようになっています。

手順

1. 新規プロジェクトを作成します。

   ```
   $ oc new-project sdn-test
   ```

2. httpd アプリケーションをデプロイします。

   ```
   $ oc new-app centos/httpd-24-
   centos7~https://github.com/openshift/httpd-ex
   ```

   ビルドが完了するまで待機します。

```
$ oc get pods
NAME                READY     STATUS      RESTARTS   AGE
httpd-ex-1-205hz    1/1       Running     0          34s
httpd-ex-1-build    0/1       Completed   0          1m
```

3. 実行中の Pod に接続します。

```
$ oc rsh po/<pod-name>
```

例:

```
$ oc rsh po/httpd-ex-1-205hz
```

4. 内部レジストリーサービスの **healthz** パスを確認します。

```
$ curl -kv https://docker-
registry.default.svc.cluster.local:5000/healthz
* About to connect() to docker-registry.default.svc.cluster.locl
port 5000 (#0)
*   Trying 172.30.150.7...
* Connected to docker-registry.default.svc.cluster.local
(172.30.150.7) port 5000 (#0)
* Initializing NSS with certpath: sql:/etc/pki/nssdb
* skipping SSL peer certificate verification
* SSL connection using TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
* Server certificate:
*  subject: CN=172.30.150.7
*  start date: Nov 30 17:21:51 2017 GMT
*  expire date: Nov 30 17:21:52 2019 GMT
*  common name: 172.30.150.7
*  issuer: CN=openshift-signer@1512059618
> GET /healthz HTTP/1.1
> User-Agent: curl/7.29.0
> Host: docker-registry.default.svc.cluster.local:5000
> Accept: */*
>
< HTTP/1.1 200 OK
< Cache-Control: no-cache
< Date: Mon, 04 Dec 2017 16:26:49 GMT
< Content-Length: 0
< Content-Type: text/plain; charset=utf-8
<
* Connection #0 to host docker-registry.default.svc.cluster.local
left intact

sh-4.2$ *exit*
```

HTTP/1.1 200 OK 応答は、ノードが適切に接続されていることを示しています。

5. テストプロジェクトをクリーンアップします。

```
$ oc delete project sdn-test
project "sdn-test" deleted
```

6. ノードホストは **TCP** ポート **10250** でリッスンしています。このポートは任意のノード上のすべてのマスターがアクセスできる必要があり、モニターがクラスターにデプロイされる場合は、インフラストラクチャーノードがすべてのインスタンスのこのポートにアクセスできる必要もあります。このポートの中断された通信は以下のコマンドで検出できます。

```
$ oc get nodes
NAME                   STATUS                        AGE       VERSION
ocp-infra-node-1clj    Ready                         4d
v1.6.1+5115d708d7
ocp-infra-node-86qr    Ready                         4d
v1.6.1+5115d708d7
ocp-infra-node-g8qw    Ready                         4d
v1.6.1+5115d708d7
ocp-master-94zd        Ready,SchedulingDisabled      4d
v1.6.1+5115d708d7
ocp-master-gjkm        Ready,SchedulingDisabled      4d
v1.6.1+5115d708d7
ocp-master-wc8w        Ready,SchedulingDisabled      4d
v1.6.1+5115d708d7
ocp-node-c5dg          Ready                         4d
v1.6.1+5115d708d7
ocp-node-ghxn          Ready                         4d
v1.6.1+5115d708d7
ocp-node-w135          NotReady                      4d
v1.6.1+5115d708d7
```

上記の出力では、**ocp-node-w135** ノードのノードサービスにマスターサービスが到達できません。これは **NotReady** ステータスで表されています。

7. 最後のサービスは、通信のルートを OpenShift Container Platform クラスターで実行される適切なサービスに指定するルーターです。ルーターは ingress トラフィック用のインフラストラクチャーノードの **TCP** ポート **80** および **443** をリッスンします。ルーターを機能させる前に、DNS が設定される必要があります。

```
$ dig *.apps.example.com

; <<>> DiG 9.11.1-P3-RedHat-9.11.1-8.P3.fc27 <<>> *.apps.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 45790
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;*.apps.example.com. IN A

;; ANSWER SECTION:
*.apps.example.com. 3571 IN CNAME apps.example.com.
apps.example.com. 3561 IN A 35.xx.xx.92

;; Query time: 0 msec
;; SERVER: 127.0.0.1#53(127.0.0.1)
;; WHEN: Tue Dec 05 16:03:52 CET 2017
;; MSG SIZE  rcvd: 105
```

IP アドレス (この場合は **35.xx.xx.92**) は、ingress トラフィックをすべてのインフラストラクチャーノードに分散させるロードバランサーをポイントするはずです。ルートの機能を確認するには、レジストリーサービスを再度チェックする必要がありますが、今回はこれをクラスター外から実行します。

```
$ curl -kv https://docker-registry-default.apps.example.com/healthz
*   Trying 35.xx.xx.92...
* TCP_NODELAY set
* Connected to docker-registry-default.apps.example.com
(35.xx.xx.92) port 443 (#0)
...
< HTTP/2 200
< cache-control: no-cache
< content-type: text/plain; charset=utf-8
< content-length: 0
< date: Tue, 05 Dec 2017 15:13:27 GMT
<
* Connection #0 to host docker-registry-default.apps.example.com
left intact
```

3.6. ストレージ

マスターインスタンスでは、**/var** ディレクトリーに 40 GB 以上のディスク容量が必要です。**df** コマンドを使用してマスターホストのディスク使用量を確認します。

```
$ df -hT
Filesystem     Type      Size  Used Avail Use% Mounted on
/dev/sda1      xfs        45G  2.8G   43G   7% /
devtmpfs       devtmpfs  3.6G     0  3.6G   0% /dev
tmpfs          tmpfs     3.6G     0  3.6G   0% /dev/shm
tmpfs          tmpfs     3.6G   63M  3.6G   2% /run
tmpfs          tmpfs     3.6G     0  3.6G   0% /sys/fs/cgroup
tmpfs          tmpfs     732M     0  732M   0% /run/user/1000
tmpfs          tmpfs     732M     0  732M   0% /run/user/0
```

ノードインスタンスでは **/var** ディレクトリーに 15 GB 以上を、Docker ストレージ (この場合は **/var/lib/docker**) にさらに 15 GB 以上が必要です。クラスターのサイズや Pod に必要な一時的なストレージの容量に応じて、別のパーティションをノード上の **/var/lib/origin/openshift.local.volumes** に作成する必要があります。

```
$ df -hT
Filesystem     Type      Size  Used Avail Use% Mounted on
/dev/sda1      xfs        25G  2.4G   23G  10% /
devtmpfs       devtmpfs  3.6G     0  3.6G   0% /dev
tmpfs          tmpfs     3.6G     0  3.6G   0% /dev/shm
tmpfs          tmpfs     3.6G  147M  3.5G   4% /run
tmpfs          tmpfs     3.6G     0  3.6G   0% /sys/fs/cgroup
/dev/sdb       xfs        25G  2.7G   23G  11% /var/lib/docker
/dev/sdc       xfs        50G   33M   50G   1%
/var/lib/origin/openshift.local.volumes
tmpfs          tmpfs     732M     0  732M   0% /run/user/1000
```

Pod の永続ストレージは OpenShift Container Platform クラスターを実行するインスタンス以外で処理される必要があります。Pod の永続ボリュームはインフラストラクチャープロバイダーによってプロビ

ジョニングされるか、または Container Native Storage または Container Ready Storage を使用してプロビジョニングできます。

3.7. DOCKER ストレージ

Docker ストレージは 2 つのオプションのどちらかでサポートされます。1 つ目のオプションはデバイスマッパーを使用したシンプール論理ボリュームで、2 つ目のオプションは overlay2 ファイルシステム (Red Hat Enterprise Linux バージョン 7.4 以降) です。通常はセットアップが容易でパフォーマンスが強化されるので、overlay2 ファイルシステムが推奨されます。

Docker ストレージディスクは **/var/lib/docker** としてマウントされ、**xfs** ファイルシステムでフォーマットされます。Docker ストレージは overlay2 ファイルシステムを使用するように設定されます。

```
$ cat /etc/sysconfig/docker-storage
DOCKER_STORAGE_OPTIONS='--storage-driver overlay2'
```

このストレージドライバーが Docker によって使用されることを確認するには、以下を実行します。

```
# docker info
Containers: 4
 Running: 4
 Paused: 0
 Stopped: 0
Images: 4
Server Version: 1.12.6
Storage Driver: overlay2
 Backing Filesystem: xfs
Logging Driver: journald
Cgroup Driver: systemd
Plugins:
 Volume: local
 Network: overlay host bridge null
 Authorization: rhel-push-plugin
Swarm: inactive
Runtimes: docker-runc runc
Default Runtime: docker-runc
Security Options: seccomp selinux
Kernel Version: 3.10.0-693.11.1.el7.x86_64
Operating System: Employee SKU
OSType: linux
Architecture: x86_64
Number of Docker Hooks: 3
CPUs: 2
Total Memory: 7.147 GiB
Name: ocp-infra-node-1clj
ID: T7T6:IQTG:WTUX:7BRU:5FI4:XUL5:PAAM:4SLW:NWKL:WU2V:NQOW:JPHC
Docker Root Dir: /var/lib/docker
Debug Mode (client): false
Debug Mode (server): false
Registry: https://registry.access.redhat.com/v1/
WARNING: bridge-nf-call-iptables is disabled
WARNING: bridge-nf-call-ip6tables is disabled
Insecure Registries:
```

```
 127.0.0.0/8
Registries: registry.access.redhat.com (secure),
registry.access.redhat.com (secure), docker.io (secure)
```

3.8. API サービスのステータス

OpenShift API サービスの **atomic-openshift-master-api.service** はすべてのマスターインスタンスで実行されます。サービスのステータスを確認するには、以下を実行します。

```
$ systemctl status atomic-openshift-master-api.service
● atomic-openshift-master-api.service - Atomic OpenShift Master API
   Loaded: loaded (/usr/lib/systemd/system/atomic-openshift-master-
api.service; enabled; vendor preset: disabled)
   Active: active (running) since Thu 2017-11-30 11:40:19 EST; 5 days ago
     Docs: https://github.com/openshift/origin
 Main PID: 30047 (openshift)
   Memory: 65.0M
   CGroup: /system.slice/atomic-openshift-master-api.service
           └─30047 /usr/bin/openshift start master api --
config=/etc/origin/master/ma...

Dec 06 09:15:49 ocp-master-94zd atomic-openshift-master-api[30047]: I1206
09:15:49.85...
Dec 06 09:15:50 ocp-master-94zd atomic-openshift-master-api[30047]: I1206
09:15:50.96...
Dec 06 09:15:52 ocp-master-94zd atomic-openshift-master-api[30047]: I1206
09:15:52.34...
```

API サービスは、以下を使用して外部でクエリーできるヘルスチェックを公開します。

```
$ curl -k https://master.example.com/healthz
ok
```

3.9. コントローラーロールの検証

OpenShift Container Platform コントローラーサービスの **atomic-openshift-master-controllers.service** はすべてのマスターホストで利用できます。このサービスはアクティブ/パッシブモードで実行され、常に 1 つのマスターでのみ実行されます。

OpenShift Container Platform コントローラーは、このサービスを実行するホストを選択する手順を実行します。現在実行されている値は、**kube-system** プロジェクトに保存される特殊な **configmap** のアノテーションに保存されます。

cluster-admin ユーザーとして、**atomic-openshift-master-controllers** サービスを実行するマスターホストを確認します。

```
$ oc get -n kube-system cm openshift-master-controllers -o yaml
apiVersion: v1
kind: ConfigMap
metadata:
  annotations:
    control-plane.alpha.kubernetes.io/leader: '{"holderIdentity":"master-
ose-master-0.example.com-10.19.115.212-
```

```
dnwrtcl4","leaseDurationSeconds":15,"acquireTime":"2018-02-
17T18:16:54Z","renewTime":"2018-02-19T13:50:33Z","leaderTransitions":16}'
  creationTimestamp: 2018-02-02T10:30:04Z
  name: openshift-master-controllers
  namespace: kube-system
  resourceVersion: "17349662"
  selfLink: /api/v1/namespaces/kube-system/configmaps/openshift-master-
controllers
  uid: 08636843-0804-11e8-8580-fa163eb934f0
```

コマンドは、以下のように **control-plane.alpha.kubernetes.io/leader** アノテーションの **holderIdentity** プロパティー内に現在のマスターコントローラーを出力します。

```
master-<hostname>-<ip>-<8_random_characters>
```

以下のコマンドを使用して出力をフィルターし、マスターホストのホスト名を検索します。

```
$ oc get -n kube-system cm openshift-master-controllers -o json | jq -r
'.metadata.annotations[] | fromjson.holderIdentity | match("^master-(.*)-
[0-9.]*-[0-9a-z]{8}$") | .captures[0].string'
ose-master-0.example.com
```

3.10. 適切な最大転送単位 (MTU) サイズの確認

最大転送単位 (MTU) を確認することにより、SSL 証明書の問題としてマスカレードを生じさせる可能性のあるネットワークの誤設定を防ぐことができます。

パケットが HTTP で送信される MTU サイズよりも大きくなる場合、物理ネットワークルーターはデータを送信するためにパケットを複数のパケットに分割できます。ただし、パケットが HTTPS で送信される MTU サイズよりも大きいと、ルーターはそのパケットのドロップを強制的に実行します。

インストールでは、以下を含む複数コンポーネントへのセキュアな通信を提供する証明書を生成します。

- マスターホスト
- ノードホスト
- インフラストラクチャーノード
- レジストリー
- ルーター

これらの証明書は、マスターノードの場合は **/etc/origin/master** ディレクトリーに、インフラおよびアプリケーションノード場合は **/etc/origin/node** ディレクトリーに配置されています。

インストール後に、「ネットワーク接続」のセクションに説明されているプロセスを使用して **REGISTRY_OPENSHIFT_SERVER_ADDR** への接続を確認できます。

前提条件

1. マスターホストから HTTPS アドレスを取得します。

```
$ oc get dc docker-registry -o
```

```
jsonpath='{.spec.template.spec.containers[].env[?
(@.name=="OPENSHIFT_DEFAULT_REGISTRY")].value}{"\n"}'
docker-registry.default.svc:5000
```

上記により、**docker-registry.default.svc:5000** の出力が生成されます。

2. **/healthz** を上記で指定される値に追加し、これを使用してすべてのホスト (マスター、インフラストラクチャー、ノード) で確認します。

```
$ curl -v https://docker-registry.default.svc:5000/healthz
* About to connect() to docker-registry.default.svc port 5000 (#0)
*   Trying 172.30.11.171...
* Connected to docker-registry.default.svc (172.30.11.171) port 5000
(#0)
* Initializing NSS with certpath: sql:/etc/pki/nssdb
*   CAfile: /etc/pki/tls/certs/ca-bundle.crt
  CApath: none
* SSL connection using TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
* Server certificate:
*  subject: CN=172.30.11.171
*  start date: Oct 18 05:30:10 2017 GMT
*  expire date: Oct 18 05:30:11 2019 GMT
*  common name: 172.30.11.171
*  issuer: CN=openshift-signer@1508303629
> GET /healthz HTTP/1.1
> User-Agent: curl/7.29.0
> Host: docker-registry.default.svc:5000
> Accept: */*
>
< HTTP/1.1 200 OK
< Cache-Control: no-cache
< Date: Tue, 24 Oct 2017 19:42:35 GMT
< Content-Length: 0
< Content-Type: text/plain; charset=utf-8
<
* Connection #0 to host docker-registry.default.svc left intact
```

上記の出力例は、SSL 接続が正常であることを確認するために使用されている MTU サイズを示しています。接続の試行が正常に実行されると接続が確立し、証明書パスと **docker-registry** に関するすべてのサーバー証明書情報を使った NSS の初期化が完了します。

MTU サイズが不適切に設定されているとタイムアウトが生じます。

```
$ curl -v https://docker-registry.default.svc:5000/healthz
* About to connect() to docker-registry.default.svc port 5000 (#0)
*   Trying 172.30.11.171...
* Connected to docker-registry.default.svc (172.30.11.171) port 5000
(#0)
* Initializing NSS with certpath: sql:/etc/pki/nssdb
```

上記の例では、接続が確立されていますが、証明書パスが指定された NSS の初期化を完了できません。この問題では、適切な「ノード設定マップ」に不適切な MTU サイズが設定されています。

この問題を解決するには、OpenShift SDN Ethernet デバイスが使用する MTU サイズよりも 50 バイト小さい値に、ノード設定マップ内で MTU サイズを調節します。

3. 必要なイーサネットデバイスの MTU サイズを表示します (例: **eth0**)。

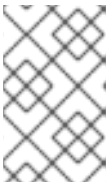
```
$ ip link show eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast
state UP mode DEFAULT qlen 1000
    link/ether fa:16:3e:92:6a:86 brd ff:ff:ff:ff:ff:ff
```

上記は MTU が 1500 に設定されていることを示しています。

4. MTU サイズを変更するには、適切な「ノード設定マップ」を修正し、**ip** コマンドの出力よりも 50 バイト少ない値に設定します。
たとえば、MTU サイズが 1500 の場合は、ノード設定マップ内で MTU サイズを 1450 に調節します。

```
networkConfig:
   mtu: 1450
```

5. 変更を保存し、ノードを再起動します。



注記

OpenShift Container Platform SDN を構成するすべてのマスターおよび ノードで MTU サイズを変更する必要があります。また、tun0 インターフェースの MTU サイズはクラスターを構成するすべてのノードで同一である必要があります。

6. ノードが再度オンラインになった後に、元の **curl** コマンドを再度実行して問題が存在しなくなっていることを確認します。

```
$ curl -v https://docker-registry.default.svc:5000/healthz
```

タイムアウトが持続する場合、引き続き MTU サイズを 50 バイト単位で調整し、このプロセスを繰り返します。

第4章 環境全体のバックアップの作成

環境全体のバックアップの作成には、インスタンスのクラッシュまたはデータの破損時の復元に役立つ重要なデータをコピーすることが必要になります。バックアップの作成後、それらは関連コンポーネントの新規バージョンに復元できます。

OpenShift Container Platform では、クラスター全体の **バックアップ** を作成できます。これにより、クラスターの現在の状態 (現在のクラスター設定) を別のストレージに保存できます。環境バックアップの状態には以下が含まれます。

- クラスターデータファイル
- 各マスターの etcd データ
- API オブジェクト
- レジストリーストレージ
- ボリュームストレージ

バックアップは、データの損失を防ぐために定期的に実行します。

重要

以下のプロセスでは、アプリケーションおよび OpenShift Container Platform クラスターをバックアップするための通常の方法について説明しています。ここではカスタム要件は考慮されません。クラスターの完全バックアップおよび復元手順の基本として以下の手順を使用してください。また、データ損失を防ぐために必要なすべての措置を取る必要があります。

バックアップと復元は保証されるものではなく、独自のデータは各自でバックアップしておく必要があります。

4.1. マスターホストのバックアップの作成

システム更新やアップグレード、またはその他の大きな変更など、OpenShift Container Platform インフラストラクチャーに変更を加える前に、このバックアッププロセスを実行するようにしてください。データのバックアップは、障害発生時に最新データが利用可能になるように定期的に実行します。

OpenShift Container Platform ファイル

マスターインスタンスは API、コントローラーなどの重要なサービスを実行します。`/etc/origin/master` ディレクトリーには、以下のような重要なファイルが数多く格納されています。

- 設定、API コントローラー、サービスなど
- インストールで生成される証明書
- すべてのクラウドプロバイダー関連の設定
- キーおよびその他の認証ファイル (htpasswd を使用する場合は `htpasswd` など)
- その他

ログレベルの引き上げやプロキシーの使用などのカスタマイズを OpenShift Container Platform サービスに対して行うことができます。設定ファイルは **/etc/sysconfig** ディレクトリーに保存されます。

マスターはノードでもあるため、**/etc/origin** ディレクトリー全体のバックアップを作成します。

手順

重要

各マスターノードで以下の手順を実行する必要があります。

1. マスターホストの設定ファイルのバックアップを作成します。

```
$ MYBACKUPDIR=/backup/$(hostname)/$(date +%Y%m%d)
$ sudo mkdir -p ${MYBACKUPDIR}/etc/sysconfig
$ sudo cp -aR /etc/origin ${MYBACKUPDIR}/etc
$ sudo cp -aR /etc/sysconfig/atomic-* ${MYBACKUPDIR}/etc/sysconfig/
```

注記

設定ファイルは、**/etc/sysconfig/atomic-openshift-master-api** および **/etc/sysconfig/atomic-openshift-master-controllers** ディレクトリーに保存されています。

警告

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

2. バックアップの計画時に考慮する必要のある他の重要なファイルには以下が含まれます。

ファイル	説明
/etc/cni/*	コンテナーネットワークインターフェースの設定 (使用される場合)
/etc/sysconfig/iptables	**iptables** ルールが保存される場所
/etc/sysconfig/docker-storage-setup	**container-storage-setup** コマンドの入力ファイル
/etc/sysconfig/docker	**docker** 設定ファイル
/etc/sysconfig/docker-network	**docker** ネットワーク設定 (例: MTU)

/etc/sysconfig/docker-storage	**docker** ストレージ設定 (**container-storage-setup** で生成される)
/etc/dnsmasq.conf	**dnsmasq** の主要な設定ファイル
/etc/dnsmasq.d/*	異なる **dnsmasq** 設定ファイル
/etc/sysconfig/flanneld	**flannel** 設定ファイル (使用される場合)
/etc/pki/ca-trust/source/anchors/	システムに追加される証明書 (例: 外部レジストリー用)

上記のファイルのバックアップを作成します。

```
$ MYBACKUPDIR=/backup/$(hostname)/$(date +%Y%m%d)
$ sudo mkdir -p ${MYBACKUPDIR}/etc/sysconfig
$ sudo mkdir -p ${MYBACKUPDIR}/etc/pki/ca-trust/source/anchors
$ sudo cp -aR /etc/sysconfig/{iptables,docker-*,flanneld} \
    ${MYBACKUPDIR}/etc/sysconfig/
$ sudo cp -aR /etc/dnsmasq* /etc/cni ${MYBACKUPDIR}/etc/
$ sudo cp -aR /etc/pki/ca-trust/source/anchors/* \
    ${MYBACKUPDIR}/etc/pki/ca-trust/source/anchors/
```

3. パッケージが間違って削除されてしまう場合や、**rpm** パッケージに含まれるファイルを復元する必要がある場合に、システムにインストールされている **rhel** パッケージの一覧があると便利です。

注記

コンテンツビューやファクトストアなどの Red Hat Satellite 機能を使用する場合は、見つからないパッケージやシステムにインストールされているパッケージの履歴データを再インストールする適切なメカニズムを指定します。

システムにインストールされている現在の **rhel** パッケージの一覧を作成するには、以下を実行します。

```
$ MYBACKUPDIR=/backup/$(hostname)/$(date +%Y%m%d)
$ sudo mkdir -p ${MYBACKUPDIR}
$ rpm -qa | sort | sudo tee $MYBACKUPDIR/packages.txt
```

4. これまでの手順を実行している場合には、以下のファイルがバックアップディレクトリーに配置されています。

```
$ MYBACKUPDIR=/backup/$(hostname)/$(date +%Y%m%d)
$ sudo find ${MYBACKUPDIR} -mindepth 1 -type f -printf '%P\n'
etc/sysconfig/atomic-openshift-master
etc/sysconfig/atomic-openshift-master-api
etc/sysconfig/atomic-openshift-master-controllers
etc/sysconfig/atomic-openshift-node
etc/sysconfig/flanneld
etc/sysconfig/iptables
```

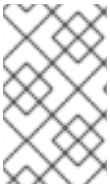
```
etc/sysconfig/docker-network
etc/sysconfig/docker-storage
etc/sysconfig/docker-storage-setup
etc/sysconfig/docker-storage-setup.rpmnew
etc/origin/master/ca.crt
etc/origin/master/ca.key
etc/origin/master/ca.serial.txt
etc/origin/master/ca-bundle.crt
etc/origin/master/master.proxy-client.crt
etc/origin/master/master.proxy-client.key
etc/origin/master/service-signer.crt
etc/origin/master/service-signer.key
etc/origin/master/serviceaccounts.private.key
etc/origin/master/serviceaccounts.public.key
etc/origin/master/openshift-master.crt
etc/origin/master/openshift-master.key
etc/origin/master/openshift-master.kubeconfig
etc/origin/master/master.server.crt
etc/origin/master/master.server.key
etc/origin/master/master.kubelet-client.crt
etc/origin/master/master.kubelet-client.key
etc/origin/master/admin.crt
etc/origin/master/admin.key
etc/origin/master/admin.kubeconfig
etc/origin/master/etcd.server.crt
etc/origin/master/etcd.server.key
etc/origin/master/master.etcd-client.key
etc/origin/master/master.etcd-client.csr
etc/origin/master/master.etcd-client.crt
etc/origin/master/master.etcd-ca.crt
etc/origin/master/policy.json
etc/origin/master/scheduler.json
etc/origin/master/htpasswd
etc/origin/master/session-secrets.yaml
etc/origin/master/openshift-router.crt
etc/origin/master/openshift-router.key
etc/origin/master/registry.crt
etc/origin/master/registry.key
etc/origin/master/master-config.yaml
etc/origin/generated-configs/master-master-
1.example.com/master.server.crt
...[OUTPUT OMITTED]...
etc/origin/cloudprovider/openstack.conf
etc/origin/node/system:node:master-0.example.com.crt
etc/origin/node/system:node:master-0.example.com.key
etc/origin/node/ca.crt
etc/origin/node/system:node:master-0.example.com.kubeconfig
etc/origin/node/server.crt
etc/origin/node/server.key
etc/origin/node/node-dnsmasq.conf
etc/origin/node/resolv.conf
etc/origin/node/node-config.yaml
etc/origin/node/flannel.etcd-client.key
etc/origin/node/flannel.etcd-client.csr
etc/origin/node/flannel.etcd-client.crt
etc/origin/node/flannel.etcd-ca.crt
```

```
etc/pki/ca-trust/source/anchors/openshift-ca.crt
etc/pki/ca-trust/source/anchors/registry-ca.crt
etc/dnsmasq.conf
etc/dnsmasq.d/origin-dns.conf
etc/dnsmasq.d/origin-upstream-dns.conf
etc/dnsmasq.d/node-dnsmasq.conf
packages.txt
```

必要な場合は、ファイルを圧縮してスペースを節約することができます。

```
$ MYBACKUPDIR=/backup/$(hostname)/$(date +%Y%m%d)
$ sudo tar -zcvf /backup/$(hostname)-$(date +%Y%m%d).tar.gz
$MYBACKUPDIR
$ sudo rm -Rf ${MYBACKUPDIR}
```

これらのファイルのいずれかをゼロから作成するには、**openshift-ansible-contrib** リポジトリーに含まれる **backup_master_node.sh** スクリプトを使用します。このスクリプトにより、ホスト上にディレクトリーが作成されます。このホストで、このスクリプトを実行して、前述のすべてのファイルをコピーします。



注記

openshift-ansible-contrib スクリプトは Red Hat ではサポートされていませんが、リファレンスアーキテクチャーチームはコードが定義通りに動作し、安全であることを確認するテストを実施しています。

このスクリプトは、以下のコマンドを使用して、全マスターホストで実行してください。

```
$ mkdir ~/git
$ cd ~/git
$ git clone https://github.com/openshift/openshift-ansible-contrib.git
$ cd openshift-ansible-contrib/reference-architecture/day2ops/scripts
$ ./backup_master_node.sh -h
```

4.2. ノードホストのバックアップの作成

ノードホストのバックアップ作成と、マスターホストのバックアップはユースケースが異なります。マスターホストには数多くの重要なファイルが含まれるため、バックアップを作成することを強く推奨します。しかしノードの性質上、フェイルオーバーが発生したときのために、特殊なアイテムはノード上で複製されて、通常は環境の実行に必要なデータは含まれません。ノードのバックアップに、環境実行に必要なアイテムが含まれる場合には、バックアップを作成することを推奨します。

システム更新やアップグレード、またはその他の大きな変更など、OpenShift Container Platform インフラストラクチャーに変更を加える前に、このバックアッププロセスを実行するようにしてください。バックアップは、障害の発生時に最新データが利用可能になるように定期的に実行する必要があります。

OpenShift Container Platform ファイル

ノードインスタンスはコンテナーをベースとする Pod の形式で実行されます。**/etc/origin/** および **/etc/origin/node** ディレクトリーは以下のような重要なファイルを格納します。

- ノードサービスの設定

- インストールで生成される証明書
- クラウドプロバイダー関連の設定
- キーおよびその他の認証ファイル (**dnsmasq** 設定など)

OpenShift Container Platform サービスは、ログレベルの引き上げやプロキシーの使用などを実行するためにカスタマイズでき、設定ファイルは **/etc/sysconfig** ディレクトリーに保存されます。

手順

1. ノード設定ファイルのバックアップを作成します。

```
$ MYBACKUPDIR=/backup/$(hostname)/$(date +%Y%m%d)
$ sudo mkdir -p ${MYBACKUPDIR}/etc/sysconfig
$ sudo cp -aR /etc/origin ${MYBACKUPDIR}/etc
$ sudo cp -aR /etc/sysconfig/atomic-openshift-node
${MYBACKUPDIR}/etc/sysconfig/
```

2. OpenShift Container Platform では以下のような特定のファイルを使用します。バックアップポリシーの計画時には、これらのファイルを考慮する必要があります。

ファイル	説明
/etc/cni/*	コンテナーネットワークインターフェースの設定 (使用される場合)
/etc/sysconfig/iptables	**iptables** ルールが保存される場所
/etc/sysconfig/docker-storage-setup	**container-storage-setup** コマンドの入力ファイル
/etc/sysconfig/docker	**docker** 設定ファイル
/etc/sysconfig/docker-network	**docker** ネットワーク設定 (例: MTU)
/etc/sysconfig/docker-storage	**docker** ストレージ設定 (**container-storage-setup** で生成される)
/etc/dnsmasq.conf	**dnsmasq** の主要な設定ファイル
/etc/dnsmasq.d/*	異なる **dnsmasq** 設定ファイル
/etc/sysconfig/flanneld	**flannel** 設定ファイル (使用される場合)
/etc/pki/ca-trust/source/anchors/	システムに追加される証明書 (例: 外部レジストリー用)

これらのファイルを作成するには、以下を実行します。

```
$ MYBACKUPDIR=/backup/$(hostname)/$(date +%Y%m%d)
$ sudo mkdir -p ${MYBACKUPDIR}/etc/sysconfig
```

```
$ sudo mkdir -p ${MYBACKUPDIR}/etc/pki/ca-trust/source/anchors
$ sudo cp -aR /etc/sysconfig/{iptables,docker-*,flanneld} \
    ${MYBACKUPDIR}/etc/sysconfig/
$ sudo cp -aR /etc/dnsmasq* /etc/cni ${MYBACKUPDIR}/etc/
$ sudo cp -aR /etc/pki/ca-trust/source/anchors/* \
    ${MYBACKUPDIR}/etc/pki/ca-trust/source/anchors/
```

3. パッケージが間違って削除されてしまう場合や、**rpm** パッケージに含まれるファイルを復元する必要がある場合に、システムにインストールされている **rhel** パッケージの一覧があると便利です。

注記

コンテンツビューやファクトストアなどの Red Hat Satellite 機能を使用する場合は、見つからないパッケージやシステムにインストールされているパッケージの履歴データを再インストールする適切なメカニズムを指定します。

システムにインストールされている現在の **rhel** パッケージの一覧を作成するには、以下を実行します。

```
$ MYBACKUPDIR=/backup/$(hostname)/$(date +%Y%m%d)
$ sudo mkdir -p ${MYBACKUPDIR}
$ rpm -qa | sort | sudo tee $MYBACKUPDIR/packages.txt
```

4. 以下のファイルがバックアップディレクトリーに配置されているはずです。

```
$ MYBACKUPDIR=/backup/$(hostname)/$(date +%Y%m%d)
$ sudo find ${MYBACKUPDIR} -mindepth 1 -type f -printf '%P\n'
etc/sysconfig/atomic-openshift-node
etc/sysconfig/flanneld
etc/sysconfig/iptables
etc/sysconfig/docker-network
etc/sysconfig/docker-storage
etc/sysconfig/docker-storage-setup
etc/sysconfig/docker-storage-setup.rpmnew
etc/origin/node/system:node:app-node-0.example.com.crt
etc/origin/node/system:node:app-node-0.example.com.key
etc/origin/node/ca.crt
etc/origin/node/system:node:app-node-0.example.com.kubeconfig
etc/origin/node/server.crt
etc/origin/node/server.key
etc/origin/node/node-dnsmasq.conf
etc/origin/node/resolv.conf
etc/origin/node/node-config.yaml
etc/origin/node/flannel.etcd-client.key
etc/origin/node/flannel.etcd-client.csr
etc/origin/node/flannel.etcd-client.crt
etc/origin/node/flannel.etcd-ca.crt
etc/origin/cloudprovider/openstack.conf
etc/pki/ca-trust/source/anchors/openshift-ca.crt
etc/pki/ca-trust/source/anchors/registry-ca.crt
etc/dnsmasq.conf
etc/dnsmasq.d/origin-dns.conf
```

```
etc/dnsmasq.d/origin-upstream-dns.conf
etc/dnsmasq.d/node-dnsmasq.conf
packages.txt
```

必要な場合は、ファイルを圧縮してスペースを節約することができます。

```
$ MYBACKUPDIR=/backup/$(hostname)/$(date +%Y%m%d)
$ sudo tar -zcvf /backup/$(hostname)-$(date +%Y%m%d).tar.gz
$MYBACKUPDIR
$ sudo rm -Rf ${MYBACKUPDIR}
```

これらのファイルのいずれかをゼロから作成するには、**openshift-ansible-contrib** リポジトリーに含まれる **backup_master_node.sh** スクリプトを使用します。このスクリプトにより、ホスト上にディレクトリーが作成されます。このホストで、このスクリプトを実行して、前述のすべてのファイルをコピーします。

注記

openshift-ansible-contrib スクリプトは Red Hat ではサポートされていませんが、リファレンスアーキテクチャーチームはコードが定義通りに動作し、安全であることを確認するテストを実施しています。

このスクリプトは、以下のコマンドを使用して、全マスターホストで実行してください。

```
$ mkdir ~/git
$ cd ~/git
$ git clone https://github.com/openshift/openshift-ansible-contrib.git
$ cd openshift-ansible-contrib/reference-architecture/day2ops/scripts
$ ./backup_master_node.sh -h
```

4.3. レジストリー証明書のバックアップ

「セキュリティー保護された外部のレジストリー」を使用する場合には、すべてのレジストリー証明書を保存する必要があります。レジストリーはデフォルトで、セキュリティー保護されています。

重要

各クラスターノードで以下の手順を実行する必要があります。

手順

1. レジストリー証明書をバックアップします。

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

2. バックアップを外部の場所に移動します。

注記

「セキュリティー保護された外部のレジストリー」を1 つ以上使用している場合には、イメージをプルまたはプッシュするホストは、Pod を実行するにはレジストリー証明書を信頼する必要があります。

4.4. 他のインストールファイルのバックアップ

OpenShift Container Platform をインストールするために使用したファイルをバックアップします。

手順

1. 復元手順には完全な再インストールが必要になるため、初期インストールで使用されたすべてのファイルを保存します。これらのファイルには、以下が含まれる場合があります。
   - (「クラスターのインストール」の場合) Ansible Playbook およびインベントリーファイル
   - (「非接続インストール」の場合) **/etc/yum.repos.d/ose.repo**
2. インストール後のステップの手順をバックアップします。一部のインストールには、インストーラーに含まれないステップが必要になる場合があります。これには、OpenShift Container Platform の制御範囲外のサービスの変更やモニターエージェントなどの追加サービスのインストールが含まれる場合があります。複数の認証プロバイダーの使用など、通常インストーラー (Advanced Installer) でサポートされていない追加の設定も必要になる場合があります。

4.5. アプリケーションデータのバックアップ

rsync がコンテナーイメージ内にインストールされていることを前提とすると、多くの場合にはアプリケーションデータは **oc rsync** コマンドを使用してバックアップできます。Red Hat **rhel7** ベースイメージには **rsync** が含まれるので、**rhel7** をベースとするすべてのイメージにはこれが含まれることになります。「CLI 操作のトラブルシューティングおよびデバッグ - rsync」を参照してください。

警告

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

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

バックアップのパスも **アプリケーションに固有** のものです。**deploymentconfig** でボリュームの **mountPath** を参照してバックアップするパスを判別することができます。

注記

この種のアプリケーションデータのバックアップは、アプリケーション Pod が実行中の場合にのみ実行できます。

手順

Jenkins デプロイメントのアプリケーションデータのバックアップ例

1. アプリケーションデータ **mountPath** を **deploymentconfig** から取得します。

```
$ oc get dc/jenkins -o jsonpath='{ .spec.template.spec.containers[?
(@.name=="jenkins")].volumeMounts[?(@.name=="jenkins-
data")].mountPath }'
/var/lib/jenkins
```

2. 現在実行中の Pod の名前を取得します。

```
$ oc get pod --selector=deploymentconfig=jenkins -o jsonpath='{
.metadata.name }'
jenkins-1-37nux
```

3. **oc rsync** コマンドを使用してアプリケーションデータをコピーします。

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

4.6. ETCD のバックアップ

etcd はすべてのオブジェクト定義、および永続マスターの状態を保存するキー値のストアです。他のコンポーネントは変更の有無を監視して、それぞれ任意の状態に切り替えます。

3.5 よりも前の OpenShift Container Platform バージョンは etcd バージョン 2 (v2) を使用し、3.5 以降ではバージョン 3 (v3) を使用します。etcd のデータモデルは、この 2 つのバージョン間で異なります。etcd v3 は v2 と v3 データモデルの両方を使用できますが、etcd v2 は v2 データモデルしか使用できません。etcd v3 サーバーでは、v2 および v3 データストアは並列して存在し、それぞれ独立しています。

v2 および v3 の両方の操作については、**ETCDCTL_API** 環境変数を使用して適切な API を使用できます。

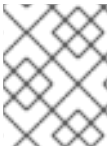
```
$ etcdctl -v
etcdctl version: 3.2.5
API version: 2
$ ETCDCTL_API=3 etcdctl version
etcdctl version: 3.2.5
API version: 3.2
```

v3 への移行方法についての詳細は、OpenShift Container Platform 3.7 ドキュメントの「Migrating etcd Data (v2 to v3)」のセクションを参照してください。

etcd のバックアッププロセスは 2 つの異なる手順で構成されています。

- 設定のバックアップ: 必要な etcd 設定および証明書が含まれます。
- データのバックアップ: v2 と v3 の両方のデータモデルが含まれます。

データのバックアッププロセスは、適切な証明書が提供され、**etcdctl** ツールがインストールされている etcd クラスターに接続できるホストで実行できます。



注記

バックアップファイルは可能な場合は OpenShift Container Platform 環境外の外部システムにコピーしてから暗号化する必要があります。

etcd のバックアップには現在のストレージボリュームへのすべての参照が含まれることに注意してください。OpenShift Container Platform は、etcd の復元時に、ノードで以前の Pod を起動して同じストレージを再割当てし始めます。このプロセスは、ノードをクラスターから削除し、新規ノードを代わりに追加するプロセスと変わりがありません。対象のノードに割り当てられているものはすべて、Pod の再スケジュール先のノードに関係なく この Pod に再び割り当てられます。

4.6.1. etcd のバックアップ

etcd のバックアップ時に、etcd 設定ファイルと etcd データの両方をバックアップする必要があります。

4.6.1.1. etcd 設定ファイルのバックアップ

保持する etcd 設定ファイルはすべて etcd が実行されているインスタンスの **/etc/etcd** ディレクトリーに保存されます。これには、etcd 設定ファイル (**/etc/etcd/etcd.conf**) およびクラスターの通信の必要な証明書が含まれます。それらすべてのファイルは Ansible インストーラーによってインストール時に生成されます。

手順
クラスターの各 etcd メンバーについての etcd 設定をバックアップします。

```
$ ssh master-0
# mkdir -p /backup/etcd-config-$(date +%Y%m%d)/
# cp -R /etc/etcd/ /backup/etcd-config-$(date +%Y%m%d)/
```

注記

各 etcd クラスターメンバーの証明書および設定ファルは一意のものです。

4.6.1.2. etcd データのバックアップ

前提条件

注記

OpenShift Container Platform インストーラーはエイリアスを作成するため、**etcdctl2** (etcd v2 タスクの場合) と **etcdctl3** (etcd v3 タスクの場合) という名前のすべてのフラグを入力しなくて済みます。

ただし、**etcdctl3** エイリアスは **etcdctl** コマンドの詳細なエンドポイント一覧を提供しないため、すべてのエンドポイントと共に **--endpoints** オプションを指定する必要があります。

etcd をバックアップする前に、以下を確認してください。

- **etcdctl** バイナリーが利用可能であるか、またはコンテナー化インストールでは **rhel7/etcd** コンテナーが利用可能であること。

- etcd クラスターとの接続を確認する (ポート 2379/tcp) こと。
- etcd クラスターに接続するための適切な証明書があることを確認すること。

1. etcd クラスターが機能していることを確認するには、その健全性を確認します。
   - etcd v2 API を使用する場合、以下のコマンドを実行します。

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

   - etcd v3 API を使用する場合、以下のコマンドを実行します。

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

2. メンバーの一覧を確認します。
   - etcd v2 API を使用する場合、以下のコマンドを実行します。

```
# etcdctl2 member list
2a371dd20f21ca8d: name=master-1.example.com
peerURLs=https://192.168.55.12:2380
clientURLs=https://192.168.55.12:2379 isLeader=false
40bef1f6c79b3163: name=master-0.example.com
peerURLs=https://192.168.55.8:2380
clientURLs=https://192.168.55.8:2379 isLeader=false
95dc17ffcce8ee29: name=master-2.example.com
peerURLs=https://192.168.55.13:2380
clientURLs=https://192.168.55.13:2379 isLeader=true
```

   - etcd v3 API を使用する場合、以下のコマンドを実行します。

```
# etcdctl3 member list
2a371dd20f21ca8d, started, master-1.example.com,
https://192.168.55.12:2380, https://192.168.55.12:2379
40bef1f6c79b3163, started, master-0.example.com,
https://192.168.55.8:2380, https://192.168.55.8:2379
95dc17ffcce8ee29, started, master-2.example.com,
https://192.168.55.13:2380, https://192.168.55.13:2379
```

手順

注記

etcdctl backup コマンドはバックアップを実行するために使用されますが、etcd v3 には **バックアップ** の概念がありません。代わりに **etcdctl snapshot save** コマンドを使用してライブメンバーの **スナップショット** を取るか、または etcd データディレクトリーの **member/snap/db** ファイルをコピーしてください。

etcdctl backup コマンドは、ノード ID やクラスター ID などのバックアップに含まれるメタデータの一部を書き換えるので、バックアップでは、ノードの以前のアイデンティティーが失われます。バックアップからクラスターを再作成するには、新規の単一ノードクラスターを作成してから、残りのノードをクラスターに追加します。メタデータは新規ノードが既存クラスターに加わらないように再作成されます。

etcd データをバックアップします。

- v2 API を使用する場合には、以下のアクションを実行してください。

  a. すべての etcd サービスを停止します。

  ```
  # systemctl stop etcd.service
  ```

  b. etcd データバックアップを作成し、etcd **db** ファイルをコピーします。

  ```
  # mkdir -p /backup/etcd-$(date +%Y%m%d)
  # etcdctl2 backup \
      --data-dir /var/lib/etcd \
      --backup-dir /backup/etcd-$(date +%Y%m%d)
  # cp /var/lib/etcd/member/snap/db /backup/etcd-$(date +%Y%m%d)
  ```

- v3 API を使用する場合、以下のコマンドを実行します。

重要

OpenShift Container Platform の以前のバージョンからアップグレードしたクラスターには、v2 データストアが含まれる可能性があるので、v2 と v3 の両方のデータストアをバックアップしてください。

```
# mkdir -p /backup/etcd-$(date +%Y%m%d)
# etcdctl3 snapshot save /backup/etcd-$(date +%Y%m%d)/db
Snapshot saved at /backup/etcd-<date>/db
# systemctl stop etcd.service
# etcdctl2 backup \
```

```
    --data-dir /var/lib/etcd \
    --backup-dir /backup/etcd-$(date +%Y%m%d)
# systemctl start etcd.service
```

注記

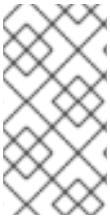
etcdctl snapshot save コマンドでは etcd サービスが実行中である必要があります。

これらのコマンドでは、**/backup/etcd-<date>/** ディレクトリーが作成されます。ここで、**<date>** は現在の日付を表します。このディレクトリーは、外部 NFS 共有、S3 バケットやその他の外部ストレージの場所のいずれかでなければなりません。

オールインワンクラスターの場合、etcd データディレクトリーは **/var/lib/origin/openshift.local.etcd** ディレクトリーに置かれます。

4.7. プロジェクトのバックアップ

関連するすべてのデータのバックアップの作成には、すべての重要な情報をエクスポートし、新規プロジェクトに復元することが関係します。



注記

OpenShift Container Platform のプロジェクトのバックアップおよび復元ツールについては、現在 Red Hat で開発中です。詳細は、以下のバグを参照してください。

- bugzilla 1303205

手順

1. バックアップ予定の関連データをすべて一覧表示します。

```
$ oc get all
NAME         TYPE      FROM      LATEST
bc/ruby-ex   Source    Git       1

NAME                TYPE      FROM          STATUS     STARTED
DURATION
builds/ruby-ex-1    Source    Git@c457001   Complete   2 minutes ago
35s

NAME                  DOCKER REPO
TAGS      UPDATED
is/guestbook          10.111.255.221:5000/myproject/guestbook
latest    2 minutes ago
is/hello-openshift    10.111.255.221:5000/myproject/hello-openshift
latest    2 minutes ago
is/ruby-22-centos7    10.111.255.221:5000/myproject/ruby-22-centos7
latest    2 minutes ago
is/ruby-ex            10.111.255.221:5000/myproject/ruby-ex
latest    2 minutes ago

NAME                  REVISION   DESIRED   CURRENT   TRIGGERED BY
dc/guestbook          1          1         1
```

```
config,image(guestbook:latest)
dc/hello-openshift   1          1         1
config,image(hello-openshift:latest)
dc/ruby-ex           1          1         1
config,image(ruby-ex:latest)

NAME                   DESIRED   CURRENT   READY     AGE
rc/guestbook-1         1         1         1         2m
rc/hello-openshift-1   1         1         1         2m
rc/ruby-ex-1           1         1         1         2m

NAME                  CLUSTER-IP       EXTERNAL-IP   PORT(S)
AGE
svc/guestbook         10.111.105.84    <none>        3000/TCP
2m
svc/hello-openshift   10.111.230.24    <none>
8080/TCP,8888/TCP   2m
svc/ruby-ex           10.111.232.117   <none>        8080/TCP
2m

NAME                          READY     STATUS      RESTARTS   AGE
po/guestbook-1-c010g          1/1       Running     0          2m
po/hello-openshift-1-4zw2q    1/1       Running     0          2m
po/ruby-ex-1-build            0/1       Completed   0          2m
po/ruby-ex-1-rxc74            1/1       Running     0          2m
```

2. プロジェクトオブジェクトを **.yaml** または **.json** ファイルにエクスポートします。

   - プロジェクトオブジェクトを **project.yaml** ファイルにエクスポートするには、以下を実行します。

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

   - プロジェクトオブジェクトを **project.json** ファイルにエクスポートするには、以下を実行します。

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

3. プロジェクトの **role bindings**、**secrets**、**service accounts**、および **persistent volume claims** をエクスポートします。

   ```
   $ for object in rolebindings serviceaccounts secrets imagestreamtags
   podpreset cms egressnetworkpolicies rolebindingrestrictions
   limitranges resourcequotas pvcs templates cronjobs statefulsets hpas
   deployments replicasets poddisruptionbudget endpoints
   do
     oc export $object -o yaml > $object.yaml
   done
   ```

4. 一部のエクスポートされたオブジェクトはプロジェクト内の特定のメタデータまたは固有の ID への参照に依存する場合があります。これは、再作成されるオブジェクトのユーザビリティーにおける制限になります。
   imagestreams の使用時に、**deploymentconfig** の **image** パラメーターは、復元される環境に存在しない内部レジストリー内のイメージの特定の **sha** チェックサムをポイントする場合

があります。たとえば、サンプル "ruby-ex" を **oc new-app centos/ruby-22-centos7~https://github.com/openshift/ruby-ex.git** として実行すると、イメージをホストするための内部レジストリーを使用する **imagestream ruby-ex** が作成されます。

```
$ oc get dc ruby-ex -o jsonpath="
{.spec.template.spec.containers[].image}"
10.111.255.221:5000/myproject/ruby-
ex@sha256:880c720b23c8d15a53b01db52f7abdcbb2280e03f686a5c8edfef1a2a7
b21cee
```

oc export でのエクスポートと同じ方法で、**deploymentconfig** をインポートすると、イメージが存在しない場合には失敗します。

このようなエクスポートを作成するには、**openshift-ansible-contrib** リポジトリーの **project_export.sh** を使用します。これにより、複数の異なるファイルに、すべてのプロジェクトオブジェクトが作成されます。このスクリプトの実行先ホストに、このプロジェクト名が指定されたディレクトリーと、そのプロジェクトに含まれるオブジェクトタイプごとの **json** ファイルが、このスクリプトにより作成されます。

注記

以下で参照される **openshift-ansible-contrib** リポジトリーのコードは Red Hat で明示的にサポートされている訳ではありませんが、リファレンスアーキテクチャーチームはコードが定義通りに動作し、安全であることを確認するためにテストを実行しています。

スクリプトは Linux で実行され、これには **jq** および **oc** コマンドがインストールされている必要があります。また、対象のプロジェクトにある全オブジェクトを読み取ることのできるユーザーで、OpenShift Container Platform 環境のシステムにログインしておく必要があります。

```
$ mkdir ~/git
$ cd ~/git
$ git clone https://github.com/openshift/openshift-ansible-
contrib.git
$ cd openshift-ansible-contrib/reference-
architecture/day2ops/scripts
$ ./project_export.sh <projectname>
```

例:

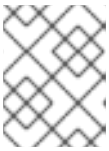
```
$ ./project_export.sh myproject
Exporting namespace to project-demo/ns.json
Exporting rolebindings to project-demo/rolebindings.json
Exporting serviceaccounts to project-demo/serviceaccounts.json
Exporting secrets to project-demo/secrets.json
Exporting deploymentconfigs to project-demo/dc_*.json
Patching DC...
Exporting buildconfigs to project-demo/bcs.json
Exporting builds to project-demo/builds.json
Exporting imagestreams to project-demo/iss.json
Exporting imagestreamtags to project-demo/imagestreamtags.json
Exporting replicationcontrollers to project-demo/rcs.json
Exporting services to project-demo/svc_*.json
```

```
Exporting pods to project-demo/pods.json
Exporting podpreset to project-demo/podpreset.json
Exporting configmaps to project-demo/cms.json
Exporting egressnetworkpolicies to project-
demo/egressnetworkpolicies.json
Exporting rolebindingrestrictions to project-
demo/rolebindingrestrictions.json
Exporting limitranges to project-demo/limitranges.json
Exporting resourcequotas to project-demo/resourcequotas.json
Exporting pvcs to project-demo/pvcs.json
Exporting routes to project-demo/routes.json
Exporting templates to project-demo/templates.json
Exporting cronjobs to project-demo/cronjobs.json
Exporting statefulsets to project-demo/statefulsets.json
Exporting hpas to project-demo/hpas.json
Exporting deployments to project-demo/deployments.json
Exporting replicasets to project-demo/replicasets.json
Exporting poddisruptionbudget to project-
demo/poddisruptionbudget.json
```

5. これが実行されたら、ファイルを確認し、コンテンツが適切にエクスポートされていることを確認します。

```
$ cd <projectname>
$ ls -1
bcs.json
builds.json
cms.json
cronjobs.json
dc_ruby-ex.json
dc_ruby-ex_patched.json
deployments.json
egressnetworkpolicies.json
endpoint_external-mysql-service.json
hpas.json
imagestreamtags.json
iss.json
limitranges.json
ns.json
poddisruptionbudget.json
podpreset.json
pods.json
pvcs.json
rcs.json
replicasets.json
resourcequotas.json
rolebindingrestrictions.json
rolebindings.json
routes.json
secrets.json
serviceaccounts.json
statefulsets.json
svc_external-mysql-service.json
svc_ruby-ex.json
```

```
templates.json
$ less bcs.json
...
```



注記

元のオブジェクトが存在しない場合、エクスポート時に空のファイルが作成されます。

6. **imagestreams** を使用している場合、スクリプトは **deploymentconfig** をイメージ **sha** の代わりにイメージ参照を使用するように変更し、**_patched** の追加情報を使用してエクスポートされたもの以外の **json** ファイルを作成します。

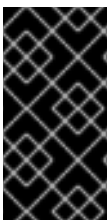
```
$ diff dc_hello-openshift.json dc_hello-openshift_patched.json
45c45
<             "image": "docker.io/openshift/hello-
openshift@sha256:42b59c869471a1b5fdacadf778667cecbaa79e002b7235f8091
540ae612f0e14",
---
>             "image": "hello-openshift:latest",
```

警告

現時点でこのスクリプトは複数のコンテナー Pod をサポートしていないため、注意して使用してください。

4.8. PERSISTENT VOLUME CLAIM (永続ボリューム要求) のバックアップ

コンテナー内の永続データをサーバーと同期できます。



重要

OpenShift Container Platform 環境をホストする一部のプロバイダーでは、バックアップおよび復元目的でサードパーティーのスナップショットサービスを起動する機能がある場合があります。ただし、OpenShift Container Platform ではこれらのサービスを起動する機能を提供していないため、本書ではこれらの手順については説明しません。

特定のアプリケーションに関する正しいバックアップ手順は、製品のドキュメントを参照してください。たとえば、mysql データディレクトリー自体をコピーしても使用可能なバックアップは作成されません。その代わりに、OpenShift Container Platform がホストするプラットフォームで提供されるスナップショットソリューションを使用するなど、関連のアプリケーション特有のバックアップ手順を実行してから、データを同期してください。

手順

1. プロジェクトおよび Pod を表示します。

```
$ oc get pods
```

```
NAME           READY     STATUS      RESTARTS   AGE
demo-1-build   0/1       Completed   0          2h
demo-2-fxx6d   1/1       Running     0          1h
```

2. 永続ボリュームで使用されているボリュームを検索できるように必要な Pod の情報を取得します。

```
$ oc describe pod demo-2-fxx6d
Name:   demo-2-fxx6d
Namespace:  test
Security Policy: restricted
Node:   ip-10-20-6-20.ec2.internal/10.20.6.20
Start Time:  Tue, 05 Dec 2017 12:54:34 -0500
Labels:   app=demo
   deployment=demo-2
   deploymentconfig=demo
Status:   Running
IP:   172.16.12.5
Controllers:  ReplicationController/demo-2
Containers:
  demo:
    Container ID:
docker://201f3e55b373641eb36945d723e1e212ecab847311109b5cee1fd010942
4217a
    Image:  docker-
registry.default.svc:5000/test/demo@sha256:0a9f2487a0d95d51511e49d20
dc9ff6f350436f935968b0c83fcb98a7a8c381a
    Image ID:  docker-pullable://docker-
registry.default.svc:5000/test/demo@sha256:0a9f2487a0d95d51511e49d20
dc9ff6f350436f935968b0c83fcb98a7a8c381a
    Port:  8080/TCP
    State:  Running
      Started:  Tue, 05 Dec 2017 12:54:52 -0500
    Ready:  True
    Restart Count: 0
    Volume Mounts:
      */opt/app-root/src/uploaded from persistent-volume (rw)*
      /var/run/secrets/kubernetes.io/serviceaccount from default-
token-8mmrk (ro)
    Environment Variables: <none>
...omitted...
```

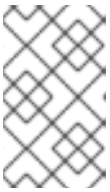
この出力は永続データが **/opt/app-root/src/uploaded** ディレクトリーにあることを示しています。

3. データをローカルにコピーします。

```
$ oc rsync demo-2-fxx6d:/opt/app-root/src/uploaded ./demo-app
receiving incremental file list
uploaded/
uploaded/ocp_sop.txt
uploaded/lost+found/

sent 38 bytes  received 190 bytes  152.00 bytes/sec
total size is 32  speedup is 0.14
```

ocp_sop.txt ファイルはローカルシステムにダウンロードされ、バックアップソフトウェアまたは別のバックアップメカニズムでバックアップされます。



注記

また、**pvc** を使用せずに Pod を起動する場合には、直前の手順を使用することもできますが、**pvc** が必要かどうかを後で確認する必要があります。データを保存してから復元プロセスを使用し、新規ストレージを生成することができます。

第5章 ホストレベルのタスク

5.1. ホストのクラスターへの追加

マスターまたはノードホストのクラスターへの追加についての詳細は、『クラスター設定ガイド』の「ホストの既存クラスターへの追加」のセクションを参照してください。

5.2. マスターホストのタスク

5.2.1. マスターホストの使用の終了

マスターホストの使用を終了することにより、マスターホストを OpenShift Container Platform 環境から削除できます。

マスターホストの使用終了やサイズ縮小が必要になる要因には、ハードウェアのサイズ変更または基礎となるインフラストラクチャーの置き換えなどが含まれます。

可用性の高い OpenShift Container Platform 環境には、少なくとも 3 つのマスターホストと 3 つの etcd ノードが必要です。通常、マスターホストは etcd サービスと同じ場所に置かれます。マスターホストの使用を終了する場合は、そのホストの etcd サービスの使用も終了する必要があります。

重要

マスターおよび etcd サービスは、サービス間で実行される投票メカニズムにより常に奇数の数でデプロイするようにします。

5.2.1.1. マスターホストのバックアップの作成

システム更新やアップグレード、またはその他の大きな変更など、OpenShift Container Platform インフラストラクチャーに変更を加える前に、このバックアッププロセスを実行するようにしてください。データのバックアップは、障害発生時に最新データが利用可能になるように定期的に実行します。

OpenShift Container Platform ファイル

マスターインスタンスは API、コントローラーなどの重要なサービスを実行します。`/etc/origin/master` ディレクトリーには、以下のような重要なファイルが数多く格納されています。

- 設定、API コントローラー、サービスなど
- インストールで生成される証明書
- すべてのクラウドプロバイダー関連の設定
- キーおよびその他の認証ファイル (htpasswd を使用する場合は `htpasswd` など)
- その他

ログレベルの引き上げやプロキシーの使用などのカスタマイズを OpenShift Container Platform サービスに対して行うことができます。設定ファイルは `/etc/sysconfig` ディレクトリーに保存されます。

マスターはノードでもあるため、`/etc/origin` ディレクトリー全体のバックアップを作成します。

手順

重要

各マスターノードで以下の手順を実行する必要があります。

1. マスターホストの設定ファイルのバックアップを作成します。

```
$ MYBACKUPDIR=/backup/$(hostname)/$(date +%Y%m%d)
$ sudo mkdir -p ${MYBACKUPDIR}/etc/sysconfig
$ sudo cp -aR /etc/origin ${MYBACKUPDIR}/etc
$ sudo cp -aR /etc/sysconfig/atomic-* ${MYBACKUPDIR}/etc/sysconfig/
```

注記

設定ファイルは、**/etc/sysconfig/atomic-openshift-master-api** および **/etc/sysconfig/atomic-openshift-master-controllers** ディレクトリーに保存されています。

警告

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

2. バックアップの計画時に考慮する必要のある他の重要なファイルには以下が含まれます。

ファイル	説明
/etc/cni/*	コンテナーネットワークインターフェースの設定 (使用される場合)
/etc/sysconfig/iptables	**iptables** ルールが保存される場所
/etc/sysconfig/docker-storage-setup	**container-storage-setup** コマンドの入力ファイル
/etc/sysconfig/docker	**docker** 設定ファイル
/etc/sysconfig/docker-network	**docker** ネットワーク設定 (例: MTU)
/etc/sysconfig/docker-storage	**docker** ストレージ設定 (**container-storage-setup** で生成される)
/etc/dnsmasq.conf	**dnsmasq** の主要な設定ファイル

/etc/dnsmasq.d/*	異なる **dnsmasq** 設定ファイル
/etc/sysconfig/flanneld	**flannel** 設定ファイル (使用される場合)
/etc/pki/ca-trust/source/anchors/	システムに追加される証明書 (例: 外部レジストリー用)

上記のファイルのバックアップを作成します。

```
$ MYBACKUPDIR=/backup/$(hostname)/$(date +%Y%m%d)
$ sudo mkdir -p ${MYBACKUPDIR}/etc/sysconfig
$ sudo mkdir -p ${MYBACKUPDIR}/etc/pki/ca-trust/source/anchors
$ sudo cp -aR /etc/sysconfig/{iptables,docker-*,flanneld} \
    ${MYBACKUPDIR}/etc/sysconfig/
$ sudo cp -aR /etc/dnsmasq* /etc/cni ${MYBACKUPDIR}/etc/
$ sudo cp -aR /etc/pki/ca-trust/source/anchors/* \
    ${MYBACKUPDIR}/etc/pki/ca-trust/source/anchors/
```

3. パッケージが間違って削除されてしまう場合や、**rpm** パッケージに含まれるファイルを復元する必要がある場合に、システムにインストールされている **rhel** パッケージの一覧があると便利です。

注記

コンテンツビューやファクトストアなどの Red Hat Satellite 機能を使用する場合は、見つからないパッケージやシステムにインストールされているパッケージの履歴データを再インストールする適切なメカニズムを指定します。

システムにインストールされている現在の **rhel** パッケージの一覧を作成するには、以下を実行します。

```
$ MYBACKUPDIR=/backup/$(hostname)/$(date +%Y%m%d)
$ sudo mkdir -p ${MYBACKUPDIR}
$ rpm -qa | sort | sudo tee $MYBACKUPDIR/packages.txt
```

4. これまでの手順を実行している場合には、以下のファイルがバックアップディレクトリーに配置されています。

```
$ MYBACKUPDIR=/backup/$(hostname)/$(date +%Y%m%d)
$ sudo find ${MYBACKUPDIR} -mindepth 1 -type f -printf '%P\n'
etc/sysconfig/atomic-openshift-master
etc/sysconfig/atomic-openshift-master-api
etc/sysconfig/atomic-openshift-master-controllers
etc/sysconfig/atomic-openshift-node
etc/sysconfig/flanneld
etc/sysconfig/iptables
etc/sysconfig/docker-network
etc/sysconfig/docker-storage
etc/sysconfig/docker-storage-setup
etc/sysconfig/docker-storage-setup.rpmnew
etc/origin/master/ca.crt
etc/origin/master/ca.key
```

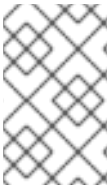
```
etc/origin/master/ca.serial.txt
etc/origin/master/ca-bundle.crt
etc/origin/master/master.proxy-client.crt
etc/origin/master/master.proxy-client.key
etc/origin/master/service-signer.crt
etc/origin/master/service-signer.key
etc/origin/master/serviceaccounts.private.key
etc/origin/master/serviceaccounts.public.key
etc/origin/master/openshift-master.crt
etc/origin/master/openshift-master.key
etc/origin/master/openshift-master.kubeconfig
etc/origin/master/master.server.crt
etc/origin/master/master.server.key
etc/origin/master/master.kubelet-client.crt
etc/origin/master/master.kubelet-client.key
etc/origin/master/admin.crt
etc/origin/master/admin.key
etc/origin/master/admin.kubeconfig
etc/origin/master/etcd.server.crt
etc/origin/master/etcd.server.key
etc/origin/master/master.etcd-client.key
etc/origin/master/master.etcd-client.csr
etc/origin/master/master.etcd-client.crt
etc/origin/master/master.etcd-ca.crt
etc/origin/master/policy.json
etc/origin/master/scheduler.json
etc/origin/master/htpasswd
etc/origin/master/session-secrets.yaml
etc/origin/master/openshift-router.crt
etc/origin/master/openshift-router.key
etc/origin/master/registry.crt
etc/origin/master/registry.key
etc/origin/master/master-config.yaml
etc/origin/generated-configs/master-master-
1.example.com/master.server.crt
...[OUTPUT OMITTED]...
etc/origin/cloudprovider/openstack.conf
etc/origin/node/system:node:master-0.example.com.crt
etc/origin/node/system:node:master-0.example.com.key
etc/origin/node/ca.crt
etc/origin/node/system:node:master-0.example.com.kubeconfig
etc/origin/node/server.crt
etc/origin/node/server.key
etc/origin/node/node-dnsmasq.conf
etc/origin/node/resolv.conf
etc/origin/node/node-config.yaml
etc/origin/node/flannel.etcd-client.key
etc/origin/node/flannel.etcd-client.csr
etc/origin/node/flannel.etcd-client.crt
etc/origin/node/flannel.etcd-ca.crt
etc/pki/ca-trust/source/anchors/openshift-ca.crt
etc/pki/ca-trust/source/anchors/registry-ca.crt
etc/dnsmasq.conf
etc/dnsmasq.d/origin-dns.conf
```

```
etc/dnsmasq.d/origin-upstream-dns.conf
etc/dnsmasq.d/node-dnsmasq.conf
packages.txt
```

必要な場合は、ファイルを圧縮してスペースを節約することができます。

```
$ MYBACKUPDIR=/backup/$(hostname)/$(date +%Y%m%d)
$ sudo tar -zcvf /backup/$(hostname)-$(date +%Y%m%d).tar.gz
$MYBACKUPDIR
$ sudo rm -Rf ${MYBACKUPDIR}
```

これらのファイルのいずれかをゼロから作成するには、**openshift-ansible-contrib** リポジトリーに含まれる **backup_master_node.sh** スクリプトを使用します。このスクリプトにより、ホスト上にディレクトリーが作成されます。このホストで、このスクリプトを実行して、前述のすべてのファイルをコピーします。



注記

openshift-ansible-contrib スクリプトは Red Hat ではサポートされていませんが、リファレンスアーキテクチャーチームはコードが定義通りに動作し、安全であることを確認するテストを実施しています。

このスクリプトは、以下のコマンドを使用して、全マスターホストで実行してください。

```
$ mkdir ~/git
$ cd ~/git
$ git clone https://github.com/openshift/openshift-ansible-contrib.git
$ cd openshift-ansible-contrib/reference-architecture/day2ops/scripts
$ ./backup_master_node.sh -h
```

5.2.1.2. etcd のバックアップ

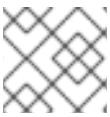
etcd のバックアップ時に、etcd 設定ファイルと etcd データの両方をバックアップする必要があります。

5.2.1.2.1. etcd 設定ファイルのバックアップ

保持する etcd 設定ファイルはすべて etcd が実行されているインスタンスの **/etc/etcd** ディレクトリーに保存されます。これには、etcd 設定ファイル (**/etc/etcd/etcd.conf**) およびクラスターの通信の必要な証明書が含まれます。それらすべてのファイルは Ansible インストーラーによってインストール時に生成されます。

手順
クラスターの各 etcd メンバーについての etcd 設定をバックアップします。

```
$ ssh master-0
# mkdir -p /backup/etcd-config-$(date +%Y%m%d)/
# cp -R /etc/etcd/ /backup/etcd-config-$(date +%Y%m%d)/
```



注記

各 etcd クラスターメンバーの証明書および設定ファルは一意のものです。

5.2.1.2.2. etcd データのバックアップ

前提条件

注記

OpenShift Container Platform インストーラーはエイリアスを作成するため、**etcdctl2** (etcd v2 タスクの場合) と **etcdctl3** (etcd v3 タスクの場合) という名前のすべてのフラグを入力しなくて済みます。

ただし、**etcdctl3** エイリアスは **etcdctl** コマンドの詳細なエンドポイント一覧を提供しないため、すべてのエンドポイントと共に **--endpoints** オプションを指定する必要があります。

etcd をバックアップする前に、以下を確認してください。

- **etcdctl** バイナリーが利用可能であるか、またはコンテナー化インストールでは **rhel7/etcd** コンテナーが利用可能であること。
- etcd クラスターとの接続を確認する (ポート 2379/tcp) こと。
- etcd クラスターに接続するための適切な証明書があることを確認すること。

手順

注記

etcdctl backup コマンドはバックアップを実行するために使用されますが、etcd v3 には **バックアップ** の概念がありません。代わりに **etcdctl snapshot save** コマンドを使用してライブメンバーの **スナップショット** を取るか、または etcd データディレクトリーの **member/snap/db** ファイルをコピーしてください。

etcdctl backup コマンドは、ノード ID やクラスター ID などのバックアップに含まれるメタデータの一部を書き換えるので、バックアップでは、ノードの以前のアイデンティティーが失われます。バックアップからクラスターを再作成するには、新規の単一ノードクラスターを作成してから、残りのノードをクラスターに追加します。メタデータは新規ノードが既存クラスターに加わらないように再作成されます。

etcd データをバックアップします。

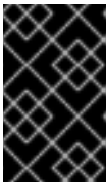
- v2 API を使用する場合には、以下のアクションを実行してください。
  a. すべての etcd サービスを停止します。

```
# systemctl stop etcd.service
```

  b. etcd データバックアップを作成し、etcd **db** ファイルをコピーします。

```
# mkdir -p /backup/etcd-$(date +%Y%m%d)
# etcdctl2 backup \
    --data-dir /var/lib/etcd \
    --backup-dir /backup/etcd-$(date +%Y%m%d)
# cp /var/lib/etcd/member/snap/db /backup/etcd-$(date +%Y%m%d)
```

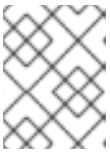
- v3 API を使用する場合、以下のコマンドを実行します。



重要

OpenShift Container Platform の以前のバージョンからアップグレードしたクラスターには、v2 データストアが含まれる可能性があるので、v2 と v3 の両方のデータストアをバックアップしてください。

```
# mkdir -p /backup/etcd-$(date +%Y%m%d)
# etcdctl3 snapshot save /backup/etcd-$(date +%Y%m%d)/db
Snapshot saved at /backup/etcd-<date>/db
# systemctl stop etcd.service
# etcdctl2 backup \
    --data-dir /var/lib/etcd \
    --backup-dir /backup/etcd-$(date +%Y%m%d)
# systemctl start etcd.service
```



注記

`etcdctl snapshot save` コマンドでは etcd サービスが実行中である必要があります。

これらのコマンドでは、**`/backup/etcd-<date>/`** ディレクトリーが作成されます。ここで、**`<date>`** は現在の日付を表します。このディレクトリーは、外部 NFS 共有、S3 バケットやその他の外部ストレージの場所のいずれかでなければなりません。

オールインワンクラスターの場合、etcd データディレクトリーは **`/var/lib/origin/openshift.local.etcd`** ディレクトリーに置かれます。

5.2.1.3. マスターホストの使用の終了

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

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

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

手順

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

```
backend mgmt8443
```

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

2. 次に、**haproxy** サービスを再起動します。

```
$ sudo systemctl restart haproxy
```

3. マスターがロードバランサーから削除された後に、API およびコントローラーサービスを無効にします。

```
$ sudo systemctl disable --now atomic-openshift-master-api
$ sudo systemctl disable --now atomic-openshift-master-controllers
```

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

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

警告

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

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

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

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

```
Port:   dns 53/UDP
Endpoints:  192.168.55.12:8053,192.168.55.13:8053
Port:   dns-tcp 53/TCP
Endpoints:  192.168.55.12:8053,192.168.55.13:8053
Session Affinity: ClientIP
Events:   <none>
```

マスターの使用が正常に終了している場合、マスターが以前に実行されていたホストを安全に削除できます。

5.2.1.4. etcd ホストの削除

復元後に etcd ホストが失敗する場合は、クラスターから削除します。

すべてのマスターホストで実行する手順

手順

1. 相互の etcd ホストを etcd クラスターから削除します。各 etcd ノードについて以下のコマンドを実行します。

```
# etcdctl -C https://<surviving host IP address>:2379 \
  --ca-file=/etc/etcd/ca.crt     \
  --cert-file=/etc/etcd/peer.crt     \
  --key-file=/etc/etcd/peer.key member remove <failed member ID>
```

2. すべてのマスターでマスター API サービスを再起動します。

```
# master-restart api restart-master controller
```

現在の etcd クラスターで実行する手順

手順

1. 失敗したホストをクラスターから削除します。

```
# etcdctl2 cluster-health
member 5ee217d19001 is healthy: got healthy result from
https://192.168.55.12:2379
member 2a529ba1840722c0 is healthy: got healthy result from
https://192.168.55.8:2379
failed to check the health of member 8372784203e11288 on
https://192.168.55.21:2379: Get https://192.168.55.21:2379/health:
dial tcp 192.168.55.21:2379: getsockopt: connection refused
member 8372784203e11288 is unreachable: [https://192.168.55.21:2379]
are all unreachable
member ed4f0efd277d7599 is healthy: got healthy result from
https://192.168.55.13:2379
cluster is healthy

# etcdctl2 member remove 8372784203e11288 (1)
Removed member 8372784203e11288 from cluster

# etcdctl2 cluster-health
member 5ee217d19001 is healthy: got healthy result from
```

```
https://192.168.55.12:2379
member 2a529ba1840722c0 is healthy: got healthy result from
https://192.168.55.8:2379
member ed4f0efd277d7599 is healthy: got healthy result from
https://192.168.55.13:2379
cluster is healthy
```

1 **remove** コマンドにはホスト名ではなく、etcd ID が必要です。

2. etcd 設定で etcd サービスの再起動時に失敗したホストを使用しないようにするには、残りのすべての etcd ホストで **/etc/etcd/etcd.conf** ファイルを変更し、**ETCD_INITIAL_CLUSTER** 変数の値から失敗したホストを削除します。

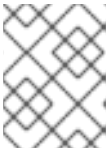
```
# vi /etc/etcd/etcd.conf
```

例:

```
ETCD_INITIAL_CLUSTER=master-
0.example.com=https://192.168.55.8:2380,master-
1.example.com=https://192.168.55.12:2380,master-
2.example.com=https://192.168.55.13:2380
```

以下のようになります。

```
ETCD_INITIAL_CLUSTER=master-
0.example.com=https://192.168.55.8:2380,master-
1.example.com=https://192.168.55.12:2380
```



注記

失敗したホストは **etcdctl** を使用して削除されているので、etcd サービスの再起動は不要です。

3. Ansible インベントリーファイルをクラスターの現在のステータスを反映し、Playbook の再実行時の問題を防げるように変更します。

```
[OSEv3:children]
masters
nodes
etcd

... [OUTPUT ABBREVIATED] ...

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

4. Flannel を使用している場合、すべてのホストの **/etc/sysconfig/flanneld** にある **flanneld** サービス設定を変更し、etcd ホストを削除します。

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

5. **flanneld** サービスを再起動します。

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

5.2.2. マスターホストのバックアップの作成

システム更新やアップグレード、またはその他の大きな変更など、OpenShift Container Platform インフラストラクチャーに変更を加える前に、このバックアッププロセスを実行するようにしてください。データのバックアップは、障害発生時に最新データが利用可能になるように定期的に実行します。

OpenShift Container Platform ファイル

マスターインスタンスは API、コントローラーなどの重要なサービスを実行します。**/etc/origin/master** ディレクトリーには、以下のような重要なファイルが数多く格納されています。

- 設定、API コントローラー、サービスなど
- インストールで生成される証明書
- すべてのクラウドプロバイダー関連の設定
- キーおよびその他の認証ファイル (htpasswd を使用する場合は **htpasswd** など)
- その他

ログレベルの引き上げやプロキシーの使用などのカスタマイズを OpenShift Container Platform サービスに対して行うことができます。設定ファイルは **/etc/sysconfig** ディレクトリーに保存されます。

マスターはノードでもあるため、**/etc/origin** ディレクトリー全体のバックアップを作成します。

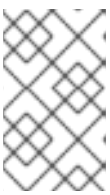
手順

重要

各マスターノードで以下の手順を実行する必要があります。

1. マスターホストの設定ファイルのバックアップを作成します。

```
$ MYBACKUPDIR=/backup/$(hostname)/$(date +%Y%m%d)
$ sudo mkdir -p ${MYBACKUPDIR}/etc/sysconfig
$ sudo cp -aR /etc/origin ${MYBACKUPDIR}/etc
$ sudo cp -aR /etc/sysconfig/atomic-* ${MYBACKUPDIR}/etc/sysconfig/
```



注記

設定ファイルは、**/etc/sysconfig/atomic-openshift-master-api** および **/etc/sysconfig/atomic-openshift-master-controllers** ディレクトリーに保存されています。

警告

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

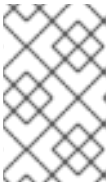
2. バックアップの計画時に考慮する必要のある他の重要なファイルには以下が含まれます。

ファイル	説明
/etc/cni/*	コンテナーネットワークインターフェースの設定 (使用される場合)
/etc/sysconfig/iptables	**iptables** ルールが保存される場所
/etc/sysconfig/docker-storage-setup	**container-storage-setup** コマンドの入力ファイル
/etc/sysconfig/docker	**docker** 設定ファイル
/etc/sysconfig/docker-network	**docker** ネットワーク設定 (例: MTU)
/etc/sysconfig/docker-storage	**docker** ストレージ設定 (**container-storage-setup** で生成される)
/etc/dnsmasq.conf	**dnsmasq** の主要な設定ファイル
/etc/dnsmasq.d/*	異なる **dnsmasq** 設定ファイル
/etc/sysconfig/flanneld	**flannel** 設定ファイル (使用される場合)
/etc/pki/ca-trust/source/anchors/	システムに追加される証明書 (例: 外部レジストリー用)

上記のファイルのバックアップを作成します。

```
$ MYBACKUPDIR=/backup/$(hostname)/$(date +%Y%m%d)
$ sudo mkdir -p ${MYBACKUPDIR}/etc/sysconfig
$ sudo mkdir -p ${MYBACKUPDIR}/etc/pki/ca-trust/source/anchors
$ sudo cp -aR /etc/sysconfig/{iptables,docker-*,flanneld} \
    ${MYBACKUPDIR}/etc/sysconfig/
$ sudo cp -aR /etc/dnsmasq* /etc/cni ${MYBACKUPDIR}/etc/
$ sudo cp -aR /etc/pki/ca-trust/source/anchors/* \
    ${MYBACKUPDIR}/etc/pki/ca-trust/source/anchors/
```

3. パッケージが間違って削除されてしまう場合や、**rpm** パッケージに含まれるファイルを復元する必要がある場合に、システムにインストールされている **rhel** パッケージの一覧があると便利です。



注記

コンテンツビューやファクトストアなどの Red Hat Satellite 機能を使用する場合は、見つからないパッケージやシステムにインストールされているパッケージの履歴データを再インストールする適切なメカニズムを指定します。

システムにインストールされている現在の **rhel** パッケージの一覧を作成するには、以下を実行します。

```
$ MYBACKUPDIR=/backup/$(hostname)/$(date +%Y%m%d)
$ sudo mkdir -p ${MYBACKUPDIR}
$ rpm -qa | sort | sudo tee $MYBACKUPDIR/packages.txt
```

4. これまでの手順を実行している場合には、以下のファイルがバックアップディレクトリーに配置されています。

```
$ MYBACKUPDIR=/backup/$(hostname)/$(date +%Y%m%d)
$ sudo find ${MYBACKUPDIR} -mindepth 1 -type f -printf '%P\n'
etc/sysconfig/atomic-openshift-master
etc/sysconfig/atomic-openshift-master-api
etc/sysconfig/atomic-openshift-master-controllers
etc/sysconfig/atomic-openshift-node
etc/sysconfig/flanneld
etc/sysconfig/iptables
etc/sysconfig/docker-network
etc/sysconfig/docker-storage
etc/sysconfig/docker-storage-setup
etc/sysconfig/docker-storage-setup.rpmnew
etc/origin/master/ca.crt
etc/origin/master/ca.key
etc/origin/master/ca.serial.txt
etc/origin/master/ca-bundle.crt
etc/origin/master/master.proxy-client.crt
etc/origin/master/master.proxy-client.key
etc/origin/master/service-signer.crt
etc/origin/master/service-signer.key
etc/origin/master/serviceaccounts.private.key
etc/origin/master/serviceaccounts.public.key
etc/origin/master/openshift-master.crt
etc/origin/master/openshift-master.key
etc/origin/master/openshift-master.kubeconfig
etc/origin/master/master.server.crt
etc/origin/master/master.server.key
etc/origin/master/master.kubelet-client.crt
etc/origin/master/master.kubelet-client.key
etc/origin/master/admin.crt
etc/origin/master/admin.key
etc/origin/master/admin.kubeconfig
etc/origin/master/etcd.server.crt
etc/origin/master/etcd.server.key
etc/origin/master/master.etcd-client.key
```

```
etc/origin/master/master.etcd-client.csr
etc/origin/master/master.etcd-client.crt
etc/origin/master/master.etcd-ca.crt
etc/origin/master/policy.json
etc/origin/master/scheduler.json
etc/origin/master/htpasswd
etc/origin/master/session-secrets.yaml
etc/origin/master/openshift-router.crt
etc/origin/master/openshift-router.key
etc/origin/master/registry.crt
etc/origin/master/registry.key
etc/origin/master/master-config.yaml
etc/origin/generated-configs/master-master-
1.example.com/master.server.crt
...[OUTPUT OMITTED]...
etc/origin/cloudprovider/openstack.conf
etc/origin/node/system:node:master-0.example.com.crt
etc/origin/node/system:node:master-0.example.com.key
etc/origin/node/ca.crt
etc/origin/node/system:node:master-0.example.com.kubeconfig
etc/origin/node/server.crt
etc/origin/node/server.key
etc/origin/node/node-dnsmasq.conf
etc/origin/node/resolv.conf
etc/origin/node/node-config.yaml
etc/origin/node/flannel.etcd-client.key
etc/origin/node/flannel.etcd-client.csr
etc/origin/node/flannel.etcd-client.crt
etc/origin/node/flannel.etcd-ca.crt
etc/pki/ca-trust/source/anchors/openshift-ca.crt
etc/pki/ca-trust/source/anchors/registry-ca.crt
etc/dnsmasq.conf
etc/dnsmasq.d/origin-dns.conf
etc/dnsmasq.d/origin-upstream-dns.conf
etc/dnsmasq.d/node-dnsmasq.conf
packages.txt
```

必要な場合は、ファイルを圧縮してスペースを節約することができます。

```
$ MYBACKUPDIR=/backup/$(hostname)/$(date +%Y%m%d)
$ sudo tar -zcvf /backup/$(hostname)-$(date +%Y%m%d).tar.gz
$MYBACKUPDIR
$ sudo rm -Rf ${MYBACKUPDIR}
```

これらのファイルのいずれかをゼロから作成するには、**openshift-ansible-contrib** リポジトリーに含まれる **backup_master_node.sh** スクリプトを使用します。このスクリプトにより、ホスト上にディレクトリーが作成されます。このホストで、このスクリプトを実行して、前述のすべてのファイルをコピーします。

注記

openshift-ansible-contrib スクリプトは Red Hat ではサポートされていませんが、リファレンスアーキテクチャーチームはコードが定義通りに動作し、安全であることを確認するテストを実施しています。

このスクリプトは、以下のコマンドを使用して、全マスターホストで実行してください。

```
$ mkdir ~/git
$ cd ~/git
$ git clone https://github.com/openshift/openshift-ansible-contrib.git
$ cd openshift-ansible-contrib/reference-architecture/day2ops/scripts
$ ./backup_master_node.sh -h
```

5.2.3. マスターホストのバックアップの復元

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

手順

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

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

警告

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

注記

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

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

   a. 現在インストールされているパッケージの一覧を取得します。

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

   b. パッケージの一覧の間に存在する差分を表示します。

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

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

c. 足りないパッケージを再インストールします。

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

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

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

```
$ MYBACKUPDIR=*/backup/$(hostname)/$(date +%Y%m%d)*
$ sudo cp ${MYBACKUPDIR}/external_certificates/my_company.crt
/etc/pki/ca-trust/source/anchors/
$ sudo update-ca-trust
```

注記

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

5.3. ノードホストのタスク

5.3.1. ノードホストの使用の終了

この使用を終了する手順は、インフラストラクチャーノードの場合でもアプリケーションノードの場合でも同じです。

前提条件
既存の Pod を削除されるノードセットから移行するために必要な容量が十分にあることを確認します。インフラストラクチャーノードの削除は、2 つ以上のノードがインフラストラクチャーノードの削除後もオンライン状態である場合にのみ推奨されます。

手順

1. 利用可能なすべてのノードを一覧表示し、使用を終了するノードを検索します。

```
$ oc get nodes
NAME                   STATUS                        AGE       VERSION
ocp-infra-node-b7pl    Ready                         23h
v1.6.1+5115d708d7
ocp-infra-node-p5zj    Ready                         23h
v1.6.1+5115d708d7
ocp-infra-node-rghb    Ready                         23h
v1.6.1+5115d708d7
ocp-master-dgf8        Ready,SchedulingDisabled      23h
v1.6.1+5115d708d7
ocp-master-q1v2        Ready,SchedulingDisabled      23h
v1.6.1+5115d708d7
ocp-master-vq70        Ready,SchedulingDisabled      23h
```

```
v1.6.1+5115d708d7
ocp-node-020m         Ready                      23h
v1.6.1+5115d708d7
ocp-node-7t5p         Ready                      23h
v1.6.1+5115d708d7
ocp-node-n0dd         Ready                      23h
v1.6.1+5115d708d7
```

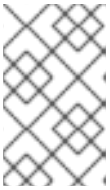
一例として、このトピックでは **ocp-infra-node-b7pl** インフラストラクチャーノードの使用を終了します。

2. ノードおよび、ノードで実行中のサービスの情報を取得します。

```
$ oc describe node ocp-infra-node-b7pl
Name:   ocp-infra-node-b7pl
Role:
Labels:   beta.kubernetes.io/arch=amd64
   beta.kubernetes.io/instance-type=n1-standard-2
   beta.kubernetes.io/os=linux
   failure-domain.beta.kubernetes.io/region=europe-west3
   failure-domain.beta.kubernetes.io/zone=europe-west3-c
   kubernetes.io/hostname=ocp-infra-node-b7pl
   role=infra
Annotations:  volumes.kubernetes.io/controller-managed-attach-
detach=true
Taints:   <none>
CreationTimestamp: Wed, 22 Nov 2017 09:36:36 -0500
Phase:
Conditions:
  ...
Addresses:  10.156.0.11,ocp-infra-node-b7pl
Capacity:
 cpu:  2
 memory: 7494480Ki
 pods:  20
Allocatable:
 cpu:  2
 memory: 7392080Ki
 pods:  20
System Info:
 Machine ID:   bc95ccf67d047f2ae42c67862c202e44
 System UUID:   9762CC3D-E23C-AB13-B8C5-FA16F0BCCE4C
 Boot ID:   ca8bf088-905d-4ec0-beec-8f89f4527ce4
 Kernel Version:  3.10.0-693.5.2.el7.x86_64
 OS Image:   Employee SKU
 Operating System:  linux
 Architecture:   amd64
 Container Runtime Version: docker://1.12.6
 Kubelet Version:  v1.6.1+5115d708d7
 Kube-Proxy Version:  v1.6.1+5115d708d7
ExternalID:   437740049672994824
Non-terminated Pods:  (2 in total)
  Namespace   Name    CPU Requests CPU Limits Memory Requests Memory
Limits
  ---------   ----    ------------ ---------- --------------- ---
----------
```

```
  default   docker-registry-1-5szjs  100m (5%) 0 (0%)  256Mi (3%)0
(0%)
  default   router-1-vzlzq   100m (5%) 0 (0%)  256Mi (3%)0 (0%)
Allocated resources:
  (Total limits may be over 100 percent, i.e., overcommitted.)
  CPU Requests CPU Limits Memory Requests Memory Limits
  ------------ ---------- --------------- -------------
  200m (10%) 0 (0%)  512Mi (7%) 0 (0%)
Events:  <none>
```

上記の出力ではノードが **router-1-vzlzq** と **docker-registry-1-5szjs** の 2 つの Pod を実行中であることを示しています。2 つ以上のインフラストラクチャーノードがこれらの 2 つの Pod を移行するために利用可能です。



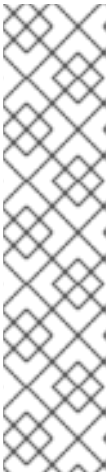
注記

上記のクラスターは可用性の高いクラスターであり、**router** と **docker-registry** の両方のサービスがすべてのインフラストラクチャーノードで実行されています。

3. ノードにスケジュール対象外のマークを付けるか、その Pod をすべて退避します。

```
$ oc adm drain ocp-infra-node-b7pl --delete-local-data
node "ocp-infra-node-b7pl" cordoned
WARNING: Deleting pods with local storage: docker-registry-1-5szjs
pod "docker-registry-1-5szjs" evicted
pod "router-1-vzlzq" evicted
node "ocp-infra-node-b7pl" drained
```

Pod に割り当て済みのローカルストレージ (**EmptyDir** など) がある場合、**--delete-local-data** オプションを指定する必要があります。通常は、実稼働で実行される Pod はローカルストレージを一時的な、またはキャッシュファイルのみに使用し、重要で永続的なファイルには使用しません。通常のストレージの場合、アプリケーションはオブジェクトストレージまたは永続ボリュームを使用します。この場合、コンテナーイメージを保存するためにオブジェクトストレージが使用されるため、**docker-registry** Pod のローカルストレージは空になります。



注記

上記の操作はノードで実行されている既存の Pod を削除します。次に、新規 Pod がレプリケーションコントローラーに応じて作成されます。

通常、すべてのアプリケーションは、レプリケーションコントローラーを使用して Pod を作成するデプロイメント設定でデプロイされる必要があります。

oc adm drain はベア Pod (Pod をミラーリングしない、または **ReplicationController**、**ReplicaSet**、**DaemonSet**、**StatefulSet**、またはジョブで管理されない Pod) を削除しません。この実行には **--force** オプションが必要です。ベア Pod は他のノードでは再作成されず、この操作中にデータが失われる可能性があることに注意してください。

以下の例は、レジストリーのレプリケーションコントローラーの出力を示しています。

```
$ oc describe rc/docker-registry-1
```

```
Name:  docker-registry-1
Namespace: default
Selector: deployment=docker-registry-1,deploymentconfig=docker-
registry,docker-registry=default
Labels:  docker-registry=default
  openshift.io/deployment-config.name=docker-registry
Annotations: ...
Replicas: 3 current / 3 desired
Pods Status: 3 Running / 0 Waiting / 0 Succeeded / 0 Failed
Pod Template:
  Labels:  deployment=docker-registry-1
   deploymentconfig=docker-registry
   docker-registry=default
  Annotations:  openshift.io/deployment-config.latest-version=1
   openshift.io/deployment-config.name=docker-registry
   openshift.io/deployment.name=docker-registry-1
  Service Account: registry
  Containers:
   registry:
    Image: openshift3/ose-docker-registry:v3.6.173.0.49
    Port: 5000/TCP
    Requests:
      cpu: 100m
      memory: 256Mi
    Liveness: http-get https://:5000/healthz delay=10s timeout=5s
period=10s #success=1 #failure=3
    Readiness: http-get https://:5000/healthz delay=0s timeout=5s
period=10s #success=1 #failure=3
    Environment:
      REGISTRY_HTTP_ADDR:     :5000
      REGISTRY_HTTP_NET:     tcp
      REGISTRY_HTTP_SECRET:
tyGEnDZmc8dQfioP3WkNd5z+Xbdfy/JVXf/NLo3s/zE=
      REGISTRY_MIDDLEWARE_REPOSITORY_OPENSHIFT_ENFORCEQUOTA: false
      REGISTRY_HTTP_TLS_KEY:     /etc/secrets/registry.key
      OPENSHIFT_DEFAULT_REGISTRY:    docker-
registry.default.svc:5000
      REGISTRY_CONFIGURATION_PATH:    /etc/registry/config.yml
      REGISTRY_HTTP_TLS_CERTIFICATE:    /etc/secrets/registry.crt
    Mounts:
      /etc/registry from docker-config (rw)
      /etc/secrets from registry-certificates (rw)
      /registry from registry-storage (rw)
  Volumes:
   registry-storage:
    Type: EmptyDir (a temporary directory that shares a pod's
lifetime)
    Medium:
   registry-certificates:
    Type: Secret (a volume populated by a Secret)
    SecretName: registry-certificates
    Optional: false
   docker-config:
    Type: Secret (a volume populated by a Secret)
    SecretName: registry-config
    Optional: false
```

```
Events:
  FirstSeen LastSeen Count From   SubObjectPath Type  Reason
Message
  --------- -------- ----- ----   ------------- -------- ------
-------
  49m  49m  1 replication-controller   Normal  SuccessfulCreate
Created pod: docker-registry-1-dprp5
```

出力の下部にあるイベントは新規 Pod 作成についての情報を表示しています。すべての Pod の一覧表示では、以下のようになります。

```
$ oc get pods
NAME                       READY     STATUS    RESTARTS   AGE
docker-registry-1-dprp5    1/1       Running   0          52m
docker-registry-1-kr8jq    1/1       Running   0          1d
docker-registry-1-ncpl2    1/1       Running   0          1d
registry-console-1-g4nqg   1/1       Running   0          1d
router-1-2gshr             0/1       Pending   0          52m
router-1-85qm4             1/1       Running   0          1d
router-1-q5sr8             1/1       Running   0          1d
```

4. 非推奨のノードで実行されていた **docker-registry-1-5szjs** および **router-1-vzlzq** Pod は、利用できなくなります。代わりに 2 つの新規 Pod **docker-registry-1-dprp5** および **router-1-2gshr** が作成されています。上記のように、新規のルーター Pod は **router-1-2gshr** ですが **Pending** 状態になります。これは、すべてのノードが単一ルーターでのみ実行でき、ホストのポート 80 および 443 にバインドされるためです。

5. 新規作成されたレジストリー Pod を確認する場合に、以下の例では、Pod が、非推奨のノードではなく、**ocp-infra-node-rghb** ノードで作成されたことが分かります。

```
$ oc describe pod docker-registry-1-dprp5
Name:   docker-registry-1-dprp5
Namespace:  default
Security Policy: hostnetwork
Node:   ocp-infra-node-rghb/10.156.0.10
...
```

アプリケーションノードとインフラストラクチャーの使用終了において、唯一の相違点は、インフラストラクチャーノードが退避された後に、そのノードを置き換える予定がない場合には、インフラストラクチャーノードで実行中のサービスをスケールダウンできる点です。

```
$ oc scale dc/router --replicas 2
deploymentconfig "router" scaled

$ oc scale dc/docker-registry --replicas 2
deploymentconfig "docker-registry" scaled
```

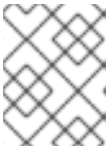
6. ここで、すべてのインフラストラクチャーノードはそれぞれの Pod を 1 種類のみ実行しています。

```
$ oc get pods
NAME                       READY     STATUS    RESTARTS   AGE
docker-registry-1-kr8jq    1/1       Running   0          1d
docker-registry-1-ncpl2    1/1       Running   0          1d
```

```
registry-console-1-g4nqg   1/1       Running   0          1d
router-1-85qm4             1/1       Running   0          1d
router-1-q5sr8             1/1       Running   0          1d

$ oc describe po/docker-registry-1-kr8jq | grep Node:
Node:   ocp-infra-node-p5zj/10.156.0.9

$ oc describe po/docker-registry-1-ncpl2 | grep Node:
Node:   ocp-infra-node-rghb/10.156.0.10
```



注記

完全に高可用のクラスターを提供するには、3 つ以上のインフラストラクチャーノードが常に利用可能である必要があります。

7. ノードのスケジューリングが無効にされていることを確認するには、以下を実行します。

```
$ oc get nodes
NAME                  STATUS                     AGE       VERSION
ocp-infra-node-b7pl   Ready,SchedulingDisabled   1d
v1.6.1+5115d708d7
ocp-infra-node-p5zj   Ready                      1d
v1.6.1+5115d708d7
ocp-infra-node-rghb   Ready                      1d
v1.6.1+5115d708d7
ocp-master-dgf8       Ready,SchedulingDisabled   1d
v1.6.1+5115d708d7
ocp-master-q1v2       Ready,SchedulingDisabled   1d
v1.6.1+5115d708d7
ocp-master-vq70       Ready,SchedulingDisabled   1d
v1.6.1+5115d708d7
ocp-node-020m         Ready                      1d
v1.6.1+5115d708d7
ocp-node-7t5p         Ready                      1d
v1.6.1+5115d708d7
ocp-node-n0dd         Ready                      1d
v1.6.1+5115d708d7
```

また、ノードに Pod が含まれていないことを確認するには、以下を実行します。

```
$ oc describe node ocp-infra-node-b7pl
Name:   ocp-infra-node-b7pl
Role:
Labels:   beta.kubernetes.io/arch=amd64
   beta.kubernetes.io/instance-type=n1-standard-2
   beta.kubernetes.io/os=linux
   failure-domain.beta.kubernetes.io/region=europe-west3
   failure-domain.beta.kubernetes.io/zone=europe-west3-c
   kubernetes.io/hostname=ocp-infra-node-b7pl
   role=infra
Annotations:  volumes.kubernetes.io/controller-managed-attach-
detach=true
Taints:   <none>
CreationTimestamp: Wed, 22 Nov 2017 09:36:36 -0500
Phase:
```

```
Conditions:
  ...
Addresses:  10.156.0.11,ocp-infra-node-b7pl
Capacity:
 cpu:  2
 memory: 7494480Ki
 pods:  20
Allocatable:
 cpu:  2
 memory: 7392080Ki
 pods:  20
System Info:
 Machine ID:   bc95ccf67d047f2ae42c67862c202e44
 System UUID:   9762CC3D-E23C-AB13-B8C5-FA16F0BCCE4C
 Boot ID:   ca8bf088-905d-4ec0-beec-8f89f4527ce4
 Kernel Version:  3.10.0-693.5.2.el7.x86_64
 OS Image:   Employee SKU
 Operating System:  linux
 Architecture:   amd64
 Container Runtime Version: docker://1.12.6
 Kubelet Version:  v1.6.1+5115d708d7
 Kube-Proxy Version:  v1.6.1+5115d708d7
ExternalID:   437740049672994824
Non-terminated Pods:  (0 in total)
  Namespace   Name  CPU Requests CPU Limits Memory Requests Memory
Limits
  ---------   ----  ------------ ---------- --------------- -----
--------
Allocated resources:
  (Total limits may be over 100 percent, i.e., overcommitted.)
  CPU Requests CPU Limits Memory Requests Memory Limits
  ------------ ---------- --------------- -------------
  0 (0%) 0 (0%)  0 (0%)  0 (0%)
Events:  <none>
```

8. インフラストラクチャーインスタンスを **/etc/haproxy/haproxy.cfg** 設定ファイルの **backend** セクションから削除します。

```
backend router80
    balance source
    mode tcp
    server infra-1.example.com 192.168.55.12:80 check
    server infra-2.example.com 192.168.55.13:80 check

backend router443
    balance source
    mode tcp
    server infra-1.example.com 192.168.55.12:443 check
    server infra-2.example.com 192.168.55.13:443 check
```

9. 次に、**haproxy** サービスを再起動します。

```
$ sudo systemctl restart haproxy
```

10. このコマンドで、すべての Pod をエビクトした後にクラスターからノードを削除します。

```
$ oc delete node ocp-infra-node-b7pl
node "ocp-infra-node-b7pl" deleted
```

```
$ oc get nodes
NAME                  STATUS                     AGE       VERSION
ocp-infra-node-p5zj   Ready                      1d
v1.6.1+5115d708d7
ocp-infra-node-rghb   Ready                      1d
v1.6.1+5115d708d7
ocp-master-dgf8       Ready,SchedulingDisabled   1d
v1.6.1+5115d708d7
ocp-master-q1v2       Ready,SchedulingDisabled   1d
v1.6.1+5115d708d7
ocp-master-vq70       Ready,SchedulingDisabled   1d
v1.6.1+5115d708d7
ocp-node-020m         Ready                      1d
v1.6.1+5115d708d7
ocp-node-7t5p         Ready                      1d
v1.6.1+5115d708d7
ocp-node-n0dd         Ready                      1d
v1.6.1+5115d708d7
```

注記

Pod またはノードの退避またはドレイン (解放) についての詳細は、「ノードの保守」のセクションを参照してください。

5.3.1.1. ノードホストの置き換え

使用終了となったノードの代わりに、ノードを追加する必要がある場合には、「ホストの既存クラスターへの追加」のセクションを参照してください。

5.3.2. ノードホストのバックアップの作成

ノードホストのバックアップ作成と、マスターホストのバックアップはユースケースが異なります。マスターホストには数多くの重要なファイルが含まれるため、バックアップを作成することを強く推奨します。しかしノードの性質上、フェイルオーバーが発生したときのために、特殊なアイテムはノード上で複製されて、通常は環境の実行に必要なデータは含まれません。ノードのバックアップに、環境実行に必要なアイテムが含まれる場合には、バックアップを作成することを推奨します。

システム更新やアップグレード、またはその他の大きな変更など、OpenShift Container Platform インフラストラクチャーに変更を加える前に、このバックアッププロセスを実行するようにしてください。バックアップは、障害の発生時に最新データが利用可能になるように定期的に実行する必要があります。

OpenShift Container Platform ファイル

ノードインスタンスはコンテナーをベースとする Pod の形式で実行されます。**`/etc/origin/`** および **`/etc/origin/node`** ディレクトリーは以下のような重要なファイルを格納します。

- ノードサービスの設定
- インストールで生成される証明書

- クラウドプロバイダー関連の設定
- キーおよびその他の認証ファイル (**dnsmasq** 設定など)

OpenShift Container Platform サービスは、ログレベルの引き上げやプロキシーの使用などを実行するためにカスタマイズでき、設定ファイルは **/etc/sysconfig** ディレクトリーに保存されます。

手順

1. ノード設定ファイルのバックアップを作成します。

```
$ MYBACKUPDIR=/backup/$(hostname)/$(date +%Y%m%d)
$ sudo mkdir -p ${MYBACKUPDIR}/etc/sysconfig
$ sudo cp -aR /etc/origin ${MYBACKUPDIR}/etc
$ sudo cp -aR /etc/sysconfig/atomic-openshift-node
${MYBACKUPDIR}/etc/sysconfig/
```

2. OpenShift Container Platform では以下のような特定のファイルを使用します。バックアップポリシーの計画時には、これらのファイルを考慮する必要があります。

ファイル	説明
/etc/cni/*	コンテナーネットワークインターフェースの設定 (使用される場合)
/etc/sysconfig/iptables	**iptables** ルールが保存される場所
/etc/sysconfig/docker-storage-setup	**container-storage-setup** コマンドの入力ファイル
/etc/sysconfig/docker	**docker** 設定ファイル
/etc/sysconfig/docker-network	**docker** ネットワーク設定 (例: MTU)
/etc/sysconfig/docker-storage	**docker** ストレージ設定 (**container-storage-setup** で生成される)
/etc/dnsmasq.conf	**dnsmasq** の主要な設定ファイル
/etc/dnsmasq.d/*	異なる **dnsmasq** 設定ファイル
/etc/sysconfig/flanneld	**flannel** 設定ファイル (使用される場合)
/etc/pki/ca-trust/source/anchors/	システムに追加される証明書 (例: 外部レジストリー用)

これらのファイルを作成するには、以下を実行します。

```
$ MYBACKUPDIR=/backup/$(hostname)/$(date +%Y%m%d)
$ sudo mkdir -p ${MYBACKUPDIR}/etc/sysconfig
$ sudo mkdir -p ${MYBACKUPDIR}/etc/pki/ca-trust/source/anchors
$ sudo cp -aR /etc/sysconfig/{iptables,docker-*,flanneld} \
```

```
    ${MYBACKUPDIR}/etc/sysconfig/
$ sudo cp -aR /etc/dnsmasq* /etc/cni ${MYBACKUPDIR}/etc/
$ sudo cp -aR /etc/pki/ca-trust/source/anchors/* \
    ${MYBACKUPDIR}/etc/pki/ca-trust/source/anchors/
```

3. パッケージが間違って削除されてしまう場合や、**rpm** パッケージに含まれるファイルを復元する必要がある場合に、システムにインストールされている **rhel** パッケージの一覧があると便利です。

注記

コンテンツビューやファクトストアなどの Red Hat Satellite 機能を使用する場合は、見つからないパッケージやシステムにインストールされているパッケージの履歴データを再インストールする適切なメカニズムを指定します。

システムにインストールされている現在の **rhel** パッケージの一覧を作成するには、以下を実行します。

```
$ MYBACKUPDIR=/backup/$(hostname)/$(date +%Y%m%d)
$ sudo mkdir -p ${MYBACKUPDIR}
$ rpm -qa | sort | sudo tee $MYBACKUPDIR/packages.txt
```

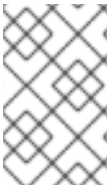
4. 以下のファイルがバックアップディレクトリーに配置されているはずです。

```
$ MYBACKUPDIR=/backup/$(hostname)/$(date +%Y%m%d)
$ sudo find ${MYBACKUPDIR} -mindepth 1 -type f -printf '%P\n'
etc/sysconfig/atomic-openshift-node
etc/sysconfig/flanneld
etc/sysconfig/iptables
etc/sysconfig/docker-network
etc/sysconfig/docker-storage
etc/sysconfig/docker-storage-setup
etc/sysconfig/docker-storage-setup.rpmnew
etc/origin/node/system:node:app-node-0.example.com.crt
etc/origin/node/system:node:app-node-0.example.com.key
etc/origin/node/ca.crt
etc/origin/node/system:node:app-node-0.example.com.kubeconfig
etc/origin/node/server.crt
etc/origin/node/server.key
etc/origin/node/node-dnsmasq.conf
etc/origin/node/resolv.conf
etc/origin/node/node-config.yaml
etc/origin/node/flannel.etcd-client.key
etc/origin/node/flannel.etcd-client.csr
etc/origin/node/flannel.etcd-client.crt
etc/origin/node/flannel.etcd-ca.crt
etc/origin/cloudprovider/openstack.conf
etc/pki/ca-trust/source/anchors/openshift-ca.crt
etc/pki/ca-trust/source/anchors/registry-ca.crt
etc/dnsmasq.conf
etc/dnsmasq.d/origin-dns.conf
etc/dnsmasq.d/origin-upstream-dns.conf
etc/dnsmasq.d/node-dnsmasq.conf
packages.txt
```

必要な場合は、ファイルを圧縮してスペースを節約することができます。

```
$ MYBACKUPDIR=/backup/$(hostname)/$(date +%Y%m%d)
$ sudo tar -zcvf /backup/$(hostname)-$(date +%Y%m%d).tar.gz
$MYBACKUPDIR
$ sudo rm -Rf ${MYBACKUPDIR}
```

これらのファイルのいずれかをゼロから作成するには、**openshift-ansible-contrib** リポジトリーに含まれる **backup_master_node.sh** スクリプトを使用します。このスクリプトにより、ホスト上にディレクトリーが作成されます。このホストで、このスクリプトを実行して、前述のすべてのファイルをコピーします。



注記

openshift-ansible-contrib スクリプトは Red Hat ではサポートされていませんが、リファレンスアーキテクチャーチームはコードが定義通りに動作し、安全であることを確認するテストを実施しています。

このスクリプトは、以下のコマンドを使用して、全マスターホストで実行してください。

```
$ mkdir ~/git
$ cd ~/git
$ git clone https://github.com/openshift/openshift-ansible-contrib.git
$ cd openshift-ansible-contrib/reference-architecture/day2ops/scripts
$ ./backup_master_node.sh -h
```

5.3.3. ノードホストバックアップの復元

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

手順

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

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

警告

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

注記

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

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

   a. 現在インストールされているパッケージの一覧を取得します。

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

   b. パッケージの一覧の間に存在する差分を表示します。

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

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

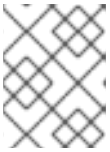
   c. 足りないパッケージを再インストールします。

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

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

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

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



注記

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

5.3.4. ノードの保守と次の手順

各種のノードの管理オプションについては、「ノードの管理」または「Pod の管理」のトピックを参照してください。ノードの管理オプションには、以下が含まれます。

- 「ノードをスケジュール対象外 (Unschedulable) またはスケジュール対象 (Schedulable) としてマークする」
- 「Pod のノードからの退避」
- 「Pod の Disruption Budget (停止状態の予算) の設定」

ノードはそのリソースの一部を特定コンポーネントによって使用されるように予約することができます。これらには、kubelet、kube-proxy、Docker、または **sshd** および **NetworkManager** などの他の残りのシステムコンポーネントが含まれます。詳細は、「ノードリソースの割り当て」を参照してください。

5.4. ETCD タスク

5.4.1. etcd のバックアップ

etcd はすべてのオブジェクト定義、および永続マスターの状態を保存するキー値のストアです。他のコンポーネントは変更の有無を監視して、それぞれ任意の状態に切り替えます。

3.5 よりも前の OpenShift Container Platform バージョンは etcd バージョン 2 (v2) を使用し、3.5 以降ではバージョン 3 (v3) を使用します。etcd のデータモデルは、この 2 つのバージョン間で異なります。etcd v3 は v2 と v3 データモデルの両方を使用できますが、etcd v2 は v2 データモデルしか使用できません。etcd v3 サーバーでは、v2 および v3 データストアは並列して存在し、それぞれ独立しています。

v2 および v3 の両方の操作については、**ETCDCTL_API** 環境変数を使用して適切な API を使用できます。

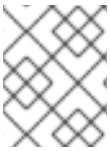
```
$ etcdctl -v
etcdctl version: 3.2.5
API version: 2
$ ETCDCTL_API=3 etcdctl version
etcdctl version: 3.2.5
API version: 3.2
```

v3 への移行方法についての詳細は、OpenShift Container Platform 3.7 ドキュメントの「Migrating etcd Data (v2 to v3)」のセクションを参照してください。

etcd のバックアッププロセスは 2 つの異なる手順で構成されています。

- 設定のバックアップ: 必要な etcd 設定および証明書が含まれます。
- データのバックアップ: v2 と v3 の両方のデータモデルが含まれます。

データのバックアッププロセスは、適切な証明書が提供され、**etcdctl** ツールがインストールされている etcd クラスターに接続できるホストで実行できます。



注記

バックアップファイルは可能な場合は OpenShift Container Platform 環境外の外部システムにコピーしてから暗号化する必要があります。

etcd のバックアップには現在のストレージボリュームへのすべての参照が含まれることに注意してください。OpenShift Container Platform は、etcd の復元時に、ノードで以前の Pod を起動して同じストレージを再割当てし始めます。このプロセスは、ノードをクラスターから削除し、新規ノードを代わりに追加するプロセスと変わりがありません。対象のノードに割り当てられているものはすべて、Pod の再スケジュール先のノードに関係なく この Pod に再び割り当てられます。

5.4.1.1. etcd のバックアップ

etcd のバックアップ時に、etcd 設定ファイルと etcd データの両方をバックアップする必要があります。

5.4.1.1.1. etcd 設定ファイルのバックアップ

保持する etcd 設定ファイルはすべて etcd が実行されているインスタンスの **/etc/etcd** ディレクト

リーに保存されます。これには、etcd 設定ファイル (**/etc/etcd/etcd.conf**) およびクラスターの通信の必要な証明書が含まれます。それらすべてのファイルは Ansible インストーラーによってインストール時に生成されます。

手順
クラスターの各 etcd メンバーについての etcd 設定をバックアップします。

```
$ ssh master-0
# mkdir -p /backup/etcd-config-$(date +%Y%m%d)/
# cp -R /etc/etcd/ /backup/etcd-config-$(date +%Y%m%d)/
```

注記

各 etcd クラスターメンバーの証明書および設定ファルは一意のものです。

5.4.1.1.2. etcd データのバックアップ

前提条件

注記

OpenShift Container Platform インストーラーはエイリアスを作成するため、**etcdctl2** (etcd v2 タスクの場合) と **etcdctl3** (etcd v3 タスクの場合) という名前のすべてのフラグを入力しなくて済みます。

ただし、**etcdctl3** エイリアスは **etcdctl** コマンドの詳細なエンドポイント一覧を提供しないため、すべてのエンドポイントと共に **--endpoints** オプションを指定する必要があります。

etcd をバックアップする前に、以下を確認してください。

- **etcdctl** バイナリーが利用可能であるか、またはコンテナー化インストールでは **rhel7/etcd** コンテナーが利用可能であること。
- etcd クラスターとの接続を確認する (ポート 2379/tcp) こと。
- etcd クラスターに接続するための適切な証明書があることを確認すること。

手順

注記

etcdctl backup コマンドはバックアップを実行するために使用されますが、etcd v3 には **バックアップ** の概念がありません。代わりに **etcdctl snapshot save** コマンドを使用してライブメンバーの **スナップショット** を取るか、または etcd データディレクトリーの **member/snap/db** ファイルをコピーしてください。

etcdctl backup コマンドは、ノード ID やクラスター ID などのバックアップに含まれるメタデータの一部を書き換えるので、バックアップでは、ノードの以前のアイデンティティーが失われます。バックアップからクラスターを再作成するには、新規の単一ノードクラスターを作成してから、残りのノードをクラスターに追加します。メタデータは新規ノードが既存クラスターに加わらないように再作成されます。

etcd データをバックアップします。

- v2 API を使用する場合には、以下のアクションを実行してください。

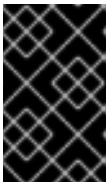
  a. すべての etcd サービスを停止します。

  ```
  # systemctl stop etcd.service
  ```

  b. etcd データバックアップを作成し、etcd **db** ファイルをコピーします。

  ```
  # mkdir -p /backup/etcd-$(date +%Y%m%d)
  # etcdctl2 backup \
      --data-dir /var/lib/etcd \
      --backup-dir /backup/etcd-$(date +%Y%m%d)
  # cp /var/lib/etcd/member/snap/db /backup/etcd-$(date +%Y%m%d)
  ```

- v3 API を使用する場合、以下のコマンドを実行します。



重要

OpenShift Container Platform の以前のバージョンからアップグレードしたクラスターには、v2 データストアが含まれる可能性があるので、v2 と v3 の両方のデータストアをバックアップしてください。

```
# mkdir -p /backup/etcd-$(date +%Y%m%d)
# etcdctl3 snapshot save /backup/etcd-$(date +%Y%m%d)/db
Snapshot saved at /backup/etcd-<date>/db
# systemctl stop etcd.service
# etcdctl2 backup \
    --data-dir /var/lib/etcd \
    --backup-dir /backup/etcd-$(date +%Y%m%d)
# systemctl start etcd.service
```

注記

`etcdctl snapshot save` コマンドでは etcd サービスが実行中である必要があります。

これらのコマンドでは、**`/backup/etcd-<date>/`** ディレクトリーが作成されます。ここで、**`<date>`** は現在の日付を表します。このディレクトリーは、外部 NFS 共有、S3 バケットやその他の外部ストレージの場所のいずれかでなければなりません。

オールインワンクラスターの場合、etcd データディレクトリーは **`/var/lib/origin/openshift.local.etcd`** ディレクトリーに置かれます。

5.4.2. etcd の復元

etcd 設定ファイルの復元手順では、適切なファイルを置き換えてからサービスを再起動します。

etcd ホストが破損し、**`/etc/etcd/etcd.conf`** ファイルが失われる場合は、以下を使用してこれを復元します。

```
$ ssh master-0
```

```
# cp /backup/yesterday/master-0-files/etcd.conf /etc/etcd/etcd.conf
# restorecon -Rv /etc/etcd/etcd.conf
# systemctl restart etcd.service
```

この例では、バックアップファイルは **/backup/yesterday/master-0-files/etcd.conf** パスに保存されており、ここでは外部 NFS 共有、S3 バケットまたはその他のストレージソリューションとして使用されます。

5.4.2.1. etcd v2 および v3 データの復元

以下のプロセスでは正常なデータファイルを復元し、etcd クラスターを単一ノードとして起動してから、etcd クラスターが必要な場合に残りのノードを追加します。

手順

1. すべての etcd サービスを停止します。

```
# systemctl stop etcd.service
```

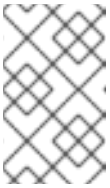
2. 適切なバックアップが復元されていることを確認するには、etcd ディレクトリーを削除します。

   - ディレクトリーを削除する前に現在の etcd データをバックアップするには、以下のコマンドを実行します。

```
# mv /var/lib/etcd /var/lib/etcd.old
# mkdir /var/lib/etcd
# chown -R etcd.etcd /var/lib/etcd/
# restorecon -Rv /var/lib/etcd/
```

   - または、ディレクトリーおよび etcd、データを削除するには、以下のコマンドを実行します。

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



注記

オールインワンクラスターの場合、etcd データディレクトリーは **/var/lib/origin/openshift.local.etcd** ディレクトリーに置かれます。

3. 正常なバックアップデータファイルをそれぞれの etcd ノードに復元します。この手順を etcd と同じ場所に配置されているマスターホストを含むすべての etcd ホストで実行します。

```
# cp -R /backup/etcd-xxx/* /var/lib/etcd/
# mv /var/lib/etcd/db /var/lib/etcd/member/snap/db
# chcon -R --reference /backup/etcd-xxx/* /var/lib/etcd/
# chown -R etcd:etcd /var/lib/etcd/R
```

4. 各ホストで etcd サービスを実行し、新規クラスターを強制的に実行します。
これにより etcd サービスのカスタムファイルが作成されます。これにより、**--force-new-cluster** オプションを追加して実行コマンドが上書きされます。

```
# mkdir -p /etc/systemd/system/etcd.service.d/
# echo "[Service]" > /etc/systemd/system/etcd.service.d/temp.conf
# echo "ExecStart=" >> /etc/systemd/system/etcd.service.d/temp.conf
# sed -n '/ExecStart/s/"$/ --force-new-cluster"/p' \
    /usr/lib/systemd/system/etcd.service \
    >> /etc/systemd/system/etcd.service.d/temp.conf

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

5. エラーメッセージの有無を確認します。

```
$ journalctl -fu etcd.service
```

6. 健全性のステータスを確認します。

```
# etcdctl2 cluster-health
member 5ee217d17301 is healthy: got healthy result from
https://192.168.55.8:2379
cluster is healthy
```

7. クラスターモードで etcd サービスを再起動します。

```
# rm -f /etc/systemd/system/etcd.service.d/temp.conf
# systemctl daemon-reload
# master-restart etcd
```

8. 健全性のステータスとメンバーの一覧を確認します。

```
# etcdctl2 cluster-health
member 5ee217d17301 is healthy: got healthy result from
https://192.168.55.8:2379
cluster is healthy

# etcdctl2 member list
5ee217d17301: name=master-0.example.com
peerURLs=http://localhost:2380 clientURLs=https://192.168.55.8:2379
isLeader=true
```

9. 最初のインスタンスの実行後に、etcd サーバーの残りの部分を復元できます。

5.4.2.1.1. peerURLS パラメーターの修正

データの復元および新規クラスターの作成後に、**peerURLs** パラメーターは、IP の代わりに etcd がピア通信をリッスンする **localhost** を示します。

```
# etcdctl2 member list
5ee217d17301: name=master-0.example.com peerURLs=http://*localhost*:2380
clientURLs=https://192.168.55.8:2379 isLeader=true
```

5.4.2.1.1.1. 手順

1. **etcdctl member list** を使用してメンバー ID を取得します。

```
`etcdctl member list`
```

2. etcd がピア通信をリッスンする IP を取得します。

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

3. メンバー情報をこの IP で更新します。

```
# etcdctl2 member update 5ee217d17301 https://192.168.55.8:2380
Updated member with ID 5ee217d17301 in cluster
```

4. 検証するには、IP がメンバーの一覧にあることを確認します。

```
$ etcdctl2 member list
5ee217d17301: name=master-0.example.com
peerURLs=https://*192.168.55.8*:2380
clientURLs=https://192.168.55.8:2379 isLeader=true
```

5.4.2.2. etcd v3 スナップショットの復元

v3 データの復元手順は v2 データの復元プロセスと同様です。

スナップショットの整合性については、復元時にオプションで検証できます。スナップショットが **etcdctl snapshot save** で作成される場合には、スナップショットに整合性ハッシュが含まれており、**etcdctl snapshot restore** でチェックされます。スナップショットがデータディレクトリーからコピーされる場合には、整合性ハッシュはなく、**--skip-hash-check** を使用しないと復元されません。

重要

v3 データのみを復元する手順は単一 etcd ホストで実行される必要があります。その後に残りのノードをクラスターに追加することができます。

手順

1. すべての etcd サービスを停止します。

```
# systemctl stop etcd.service
```

2. 古いデータすべてについては、**etcdctl** が復元手順が実行されるノードで再作成を実行するためにこれをクリアします。

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

3. **snapshot restore** コマンドを実行し、**/etc/etcd/etcd.conf** ファイルの値を置き換えます。

```
# etcdctl3 snapshot restore /backup/etcd-xxxxxx/backup.db \
  --data-dir /var/lib/etcd \
  --name master-0.example.com \
  --initial-cluster "master-0.example.com=https://192.168.55.8:2380"
\
```

```
  --initial-cluster-token "etcd-cluster-1" \
  --initial-advertise-peer-urls https://192.168.55.8:2380 \
  --skip-hash-check=true

2017-10-03 08:55:32.440779 I | mvcc: restore compact to 1041269
2017-10-03 08:55:32.468244 I | etcdserver/membership: added member
40bef1f6c79b3163 [https://192.168.55.8:2380] to cluster
26841ebcf610583c
```

4. パーミッションおよび **selinux** コンテキストを復元されたファイルに復元します。

```
# chown -R etcd.etcd /var/lib/etcd/
# restorecon -Rv /var/lib/etcd
```

5. etcd サービスを起動します。

```
# systemctl start etcd
```

6. エラーメッセージの有無を確認します。

```
# master-logs etcd etcd
```

5.4.3. etcd ホストの置き換え

etcd ホストを置き換えるには、etcd クラスターを拡張してからホストを削除します。このプロセスでは、置き換え手順の実行時に etcd ホストが失われる場合に備えてクォーラム (定足数) を維持できるようにします。

警告

etcd クラスターは置き換え操作時に クォーラム (定足数) を維持する必要があります。これは、常に 1 つ以上のホストが稼働している必要があることを意味します。

ホスト置き換え操作が etcd クラスターがクォーラム (定足数) を維持している状態で実行される場合、クラスター操作はこの影響を受けません。大規模な etcd データの複製が必要な場合には、一部の操作の速度が下がる可能性があります。

注記

etcd クラスターに関係するいずれかの手順を起動する前に、etcd データと設定ファイルのバックアップを行い、手順が失敗する際にクラスターを復元できるようにします。

5.4.4. etcd のスケーリング

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

注記

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

奇数の etcd ホストを含むクラスターの場合、フォールトトレランスに対応できます。etcd ホストが奇数分あると、クォーラム (定足数) に必要な数は変わりませんが、障害発生時の耐性が高まります。たとえば、クラスターが 3 つのメンバーで構成される場合には、クォーラム (定足数) は 2 で、残り の 1 つが耐障害性用になります。これにより、クラスターはメンバーの 2 つが正常である限り、機能し続けます。

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

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

前提条件

1. 新規 etcd ホストを追加する前に、「etcd 設定およびデータのバックアップ」を行ってデータの損失を防ぎます。
2. 現在の etcd クラスターステータスを確認し、新規ホストを正常でないクラスターに追加することを防ぎます。
   - v2 etcd api を使用する場合、以下のコマンドを使用します。

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

   - v3 etcd api を使用する場合は、以下のコマンドを実行します。

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

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

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

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

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

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

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

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

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

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

5.4.4.1. Ansible を使用した新規 etcd ホストの追加

手順

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

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

... [OUTPUT ABBREVIATED] ...

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

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

(1)(2)(3)これらの行を追加します。

2. OpenShift Container Platform がインストールされており、Ansible インベントリーファイルをホストするホストから、etcd **scaleup** Playbook を実行します。

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

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

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

... [OUTPUT ABBREVIATED] ...

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

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

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

5. **flanneld** サービスを再起動します。

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

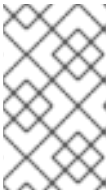
5.4.4.2. 新規 etcd ホストの手動による追加

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

1. 環境変数を作成します。

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

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



注記

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

2. 設定および証明書を保存するディレクトリーを作成します。

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

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

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

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

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

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

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

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

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

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

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

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

```
peers="https://172.18.1.18:2379,https://172.18.9.202:2379,https:/
/172.18.0.75:2379" \ 1
    member list
```

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

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

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

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

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

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

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

警告

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

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

Added member named 10.3.9.222 with ID 4e1db163a21d7651 to cluster

ETCD_NAME="<NEW_ETCD_HOSTNAME>"
ETCD_INITIAL_CLUSTER="
<NEW_ETCD_HOSTNAME>=https://<NEW_HOST_IP>:2380,
```

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

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

8. サンプル **${PREFIX}/etcd.conf** ファイルを更新します。
   a. 以下の値を直前の手順で生成された値に置き換えます。
      - ETCD_NAME
      - ETCD_INITIAL_CLUSTER
      - ETCD_INITIAL_CLUSTER_STATE
   b. 以下の変数は、直前の手順で出力された新規ホストの IP に変更します。**${NEW_ETCD_IP}** は、値として使用できます。

```
ETCD_LISTEN_PEER_URLS
ETCD_LISTEN_CLIENT_URLS
ETCD_INITIAL_ADVERTISE_PEER_URLS
ETCD_ADVERTISE_CLIENT_URLS
```

   c. メンバーシステムを etcd ノードとして使用していた場合には、**/etc/etcd/etcd.conf** ファイルの現在の値を上書きする必要があります。
   d. ファイルで構文エラーや欠落している IP アドレスがないかを確認します。エラーや欠落がある場合には、etced サービスが失敗してしまう可能性があります。

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

9. インストールファイルをホストするノードでは、**/etc/ansible/hosts** インベントリーファイルの **[etcd]** ホストグループを更新します。古い etcd ホストを削除し、新規ホストを追加します。
10. 証明書、サンプル設定ファイル、および **ca** を含む **tgz** ファイルを作成し、これを新規ホストにコピーします。

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

新規 etcd ホストを変更します。

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

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

2. etcd の通信を許可する **OS_FIREWALL_ALLOW** ファイアウォールルールを作成します。
   - クライアントのポート 2379/tcp

- ピア通信のポート 2380/tcp

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

注記

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

警告

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

3. etcd をインストールします。

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

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

4. etcd サービスが実行されていないことを確認します。

```
# systemctl disable etcd --now
```

5. etcd 設定およびデータを削除します。

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

6. 証明書および設定ファイルを展開します。

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

7. ファイル所有者のパーミッションを変更します。

```
# chown -R etcd/etcd /etc/etcd/*
# chown -R etcd/etcd /var/lib/etcd/
```

8. 新規ホストで etcd を起動します。

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

9. ホストがクラスターの一部であることと現在のクラスターの健全性を確認します。

   - v2 etcd api を使用する場合は、以下のコマンドを実行します。

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

   - v3 etcd api を使用する場合は、以下のコマンドを実行します。

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

各 OpenShift Container Platform マスターの変更

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

```
etcdClientInfo:
  ca: master.etcd-ca.crt
  certFile: master.etcd-client.crt
  keyFile: master.etcd-client.key
  urls:
```

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

2. マスター API サービスを再起動します。

   - 全マスターのインストールに対しては、以下を実行します。

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

   - または、単一マスタークラスターのインストールでは以下を実行してください。

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

     警告

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

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

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

4. **flanneld** サービスを再起動します。

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

5.4.5. etcd ホストの削除

復元後に etcd ホストが失敗する場合は、クラスターから削除します。

すべてのマスターホストで実行する手順

手順

1. 相互の etcd ホストを etcd クラスターから削除します。各 etcd ノードについて以下のコマンドを実行します。

   ```
   # etcdctl -C https://<surviving host IP address>:2379 \
     --ca-file=/etc/etcd/ca.crt     \
     --cert-file=/etc/etcd/peer.crt     \
     --key-file=/etc/etcd/peer.key member remove <failed member ID>
   ```

2. すべてのマスターでマスター API サービスを再起動します。

```
# master-restart api restart-master controller
```

現在の etcd クラスターで実行する手順

手順

1. 失敗したホストをクラスターから削除します。

```
# etcdctl2 cluster-health
member 5ee217d19001 is healthy: got healthy result from
https://192.168.55.12:2379
member 2a529ba1840722c0 is healthy: got healthy result from
https://192.168.55.8:2379
failed to check the health of member 8372784203e11288 on
https://192.168.55.21:2379: Get https://192.168.55.21:2379/health:
dial tcp 192.168.55.21:2379: getsockopt: connection refused
member 8372784203e11288 is unreachable: [https://192.168.55.21:2379]
are all unreachable
member ed4f0efd277d7599 is healthy: got healthy result from
https://192.168.55.13:2379
cluster is healthy

# etcdctl2 member remove 8372784203e11288 (1)
Removed member 8372784203e11288 from cluster

# etcdctl2 cluster-health
member 5ee217d19001 is healthy: got healthy result from
https://192.168.55.12:2379
member 2a529ba1840722c0 is healthy: got healthy result from
https://192.168.55.8:2379
member ed4f0efd277d7599 is healthy: got healthy result from
https://192.168.55.13:2379
cluster is healthy
```

(1) **remove** コマンドにはホスト名ではなく、etcd ID が必要です。

2. etcd 設定で etcd サービスの再起動時に失敗したホストを使用しないようにするには、残りのすべての etcd ホストで **/etc/etcd/etcd.conf** ファイルを変更し、**ETCD_INITIAL_CLUSTER** 変数の値から失敗したホストを削除します。

```
# vi /etc/etcd/etcd.conf
```

例:

```
ETCD_INITIAL_CLUSTER=master-
0.example.com=https://192.168.55.8:2380,master-
1.example.com=https://192.168.55.12:2380,master-
2.example.com=https://192.168.55.13:2380
```

以下のようになります。

```
ETCD_INITIAL_CLUSTER=master-
0.example.com=https://192.168.55.8:2380,master-
1.example.com=https://192.168.55.12:2380
```

注記

失敗したホストは **etcdctl** を使用して削除されているので、etcd サービスの再起動は不要です。

3. Ansible インベントリーファイルをクラスターの現在のステータスを反映し、Playbook の再実行時の問題を防げるように変更します。

```
[OSEv3:children]
masters
nodes
etcd

... [OUTPUT ABBREVIATED] ...

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

4. Flannel を使用している場合、すべてのホストの **/etc/sysconfig/flanneld** にある **flanneld** サービス設定を変更し、etcd ホストを削除します。

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

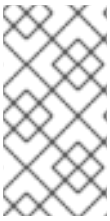
5. **flanneld** サービスを再起動します。

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

第6章 プロジェクトレベルのタスク

6.1. プロジェクトのバックアップ

関連するすべてのデータのバックアップの作成には、すべての重要な情報をエクスポートし、新規プロジェクトに復元することが関係します。



注記

OpenShift Container Platform のプロジェクトのバックアップおよび復元ツールについては、現在 Red Hat で開発中です。詳細は、以下のバグを参照してください。

- bugzilla 1303205

手順

1. バックアップ予定の関連データをすべて一覧表示します。

```
$ oc get all
NAME         TYPE      FROM      LATEST
bc/ruby-ex   Source    Git       1

NAME               TYPE      FROM          STATUS     STARTED
DURATION
builds/ruby-ex-1   Source    Git@c457001   Complete   2 minutes ago
35s

NAME                 DOCKER REPO
TAGS      UPDATED
is/guestbook         10.111.255.221:5000/myproject/guestbook
latest    2 minutes ago
is/hello-openshift   10.111.255.221:5000/myproject/hello-openshift
latest    2 minutes ago
is/ruby-22-centos7   10.111.255.221:5000/myproject/ruby-22-centos7
latest    2 minutes ago
is/ruby-ex           10.111.255.221:5000/myproject/ruby-ex
latest    2 minutes ago

NAME                 REVISION   DESIRED   CURRENT   TRIGGERED BY
dc/guestbook         1          1         1
config,image(guestbook:latest)
dc/hello-openshift   1          1         1
config,image(hello-openshift:latest)
dc/ruby-ex           1          1         1
config,image(ruby-ex:latest)

NAME                   DESIRED   CURRENT   READY     AGE
rc/guestbook-1         1         1         1         2m
rc/hello-openshift-1   1         1         1         2m
rc/ruby-ex-1           1         1         1         2m

NAME                  CLUSTER-IP       EXTERNAL-IP   PORT(S)
AGE
svc/guestbook         10.111.105.84    <none>        3000/TCP
```

```
2m
svc/hello-openshift   10.111.230.24    <none>
8080/TCP,8888/TCP   2m
svc/ruby-ex           10.111.232.117   <none>        8080/TCP
2m

NAME                          READY     STATUS      RESTARTS   AGE
po/guestbook-1-c010g          1/1       Running     0          2m
po/hello-openshift-1-4zw2q    1/1       Running     0          2m
po/ruby-ex-1-build            0/1       Completed   0          2m
po/ruby-ex-1-rxc74            1/1       Running     0          2m
```

2. プロジェクトオブジェクトを **.yaml** または **.json** ファイルにエクスポートします。

   - プロジェクトオブジェクトを **project.yaml** ファイルにエクスポートするには、以下を実行します。

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

   - プロジェクトオブジェクトを **project.json** ファイルにエクスポートするには、以下を実行します。

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

3. プロジェクトの **role bindings**、**secrets**、**service accounts**、および **persistent volume claims** をエクスポートします。

   ```
   $ for object in rolebindings serviceaccounts secrets imagestreamtags
   podpreset cms egressnetworkpolicies rolebindingrestrictions
   limitranges resourcequotas pvcs templates cronjobs statefulsets hpas
   deployments replicasets poddisruptionbudget endpoints
   do
     oc export $object -o yaml > $object.yaml
   done
   ```

4. 一部のエクスポートされたオブジェクトはプロジェクト内の特定のメタデータまたは固有の ID への参照に依存する場合があります。これは、再作成されるオブジェクトのユーザビリティーにおける制限になります。
   imagestreams の使用時に、**deploymentconfig** の **image** パラメーターは、復元される環境に存在しない内部レジストリー内のイメージの特定の **sha** チェックサムをポイントする場合があります。たとえば、サンプル "ruby-ex" を **oc new-app centos/ruby-22-centos7~https://github.com/openshift/ruby-ex.git** として実行すると、イメージをホストするための内部レジストリーを使用する **imagestream ruby-ex** が作成されます。

   ```
   $ oc get dc ruby-ex -o jsonpath="
   {.spec.template.spec.containers[].image}"
   10.111.255.221:5000/myproject/ruby-
   ex@sha256:880c720b23c8d15a53b01db52f7abdcbb2280e03f686a5c8edfef1a2a7
   b21cee
   ```

   oc export でのエクスポートと同じ方法で、**deploymentconfig** をインポートすると、イメージが存在しない場合には失敗します。

   このようなエクスポートを作成するには、**openshift-ansible-contrib** リポジトリーの

project_export.sh を使用します。これにより、複数の異なるファイルに、すべてのプロジェクトオブジェクトが作成されます。このスクリプトの実行先ホストに、このプロジェクト名が指定されたディレクトリーと、そのプロジェクトに含まれるオブジェクトタイプごとの **json** ファイルが、このスクリプトにより作成されます。

注記

以下で参照される **openshift-ansible-contrib** リポジトリーのコードは Red Hat で明示的にサポートされている訳ではありませんが、リファレンスアーキテクチャーチームはコードが定義通りに動作し、安全であることを確認するためにテストを実行しています。

スクリプトは Linux で実行され、これには **jq** および **oc** コマンドがインストールされている必要があります。また、対象のプロジェクトにある全オブジェクトを読み取ることのできるユーザーで、OpenShift Container Platform 環境のシステムにログインしておく必要があります。

```
$ mkdir ~/git
$ cd ~/git
$ git clone https://github.com/openshift/openshift-ansible-
contrib.git
$ cd openshift-ansible-contrib/reference-
architecture/day2ops/scripts
$ ./project_export.sh <projectname>
```

例:

```
$ ./project_export.sh myproject
Exporting namespace to project-demo/ns.json
Exporting rolebindings to project-demo/rolebindings.json
Exporting serviceaccounts to project-demo/serviceaccounts.json
Exporting secrets to project-demo/secrets.json
Exporting deploymentconfigs to project-demo/dc_*.json
Patching DC...
Exporting buildconfigs to project-demo/bcs.json
Exporting builds to project-demo/builds.json
Exporting imagestreams to project-demo/iss.json
Exporting imagestreamtags to project-demo/imagestreamtags.json
Exporting replicationcontrollers to project-demo/rcs.json
Exporting services to project-demo/svc_*.json
Exporting pods to project-demo/pods.json
Exporting podpreset to project-demo/podpreset.json
Exporting configmaps to project-demo/cms.json
Exporting egressnetworkpolicies to project-
demo/egressnetworkpolicies.json
Exporting rolebindingrestrictions to project-
demo/rolebindingrestrictions.json
Exporting limitranges to project-demo/limitranges.json
Exporting resourcequotas to project-demo/resourcequotas.json
Exporting pvcs to project-demo/pvcs.json
Exporting routes to project-demo/routes.json
Exporting templates to project-demo/templates.json
Exporting cronjobs to project-demo/cronjobs.json
Exporting statefulsets to project-demo/statefulsets.json
Exporting hpas to project-demo/hpas.json
```

```
Exporting deployments to project-demo/deployments.json
Exporting replicasets to project-demo/replicasets.json
Exporting poddisruptionbudget to project-
demo/poddisruptionbudget.json
```

5. これが実行されたら、ファイルを確認し、コンテンツが適切にエクスポートされていることを確認します。

```
$ cd <projectname>
$ ls -1
bcs.json
builds.json
cms.json
cronjobs.json
dc_ruby-ex.json
dc_ruby-ex_patched.json
deployments.json
egressnetworkpolicies.json
endpoint_external-mysql-service.json
hpas.json
imagestreamtags.json
iss.json
limitranges.json
ns.json
poddisruptionbudget.json
podpreset.json
pods.json
pvcs.json
rcs.json
replicasets.json
resourcequotas.json
rolebindingrestrictions.json
rolebindings.json
routes.json
secrets.json
serviceaccounts.json
statefulsets.json
svc_external-mysql-service.json
svc_ruby-ex.json
templates.json
$ less bcs.json
...
```

注記

元のオブジェクトが存在しない場合、エクスポート時に空のファイルが作成されます。

6. **imagestreams** を使用している場合、スクリプトは **deploymentconfig** をイメージ **sha** の代わりにイメージ参照を使用するように変更し、**_patched** の追加情報を使用してエクスポートされたもの以外の **json** ファイルを作成します。

```
$ diff dc_hello-openshift.json dc_hello-openshift_patched.json
45c45
```

```
<              "image": "docker.io/openshift/hello-
openshift@sha256:42b59c869471a1b5fdacadf778667cecbaa79e002b7235f8091
540ae612f0e14",
---
>              "image": "hello-openshift:latest",
```

警告

現時点でこのスクリプトは複数のコンテナー Pod をサポートしていないため、注意して使用してください。

6.2. プロジェクトの復元

プロジェクトを復元するために、**oc create -f pods.json** を実行して新規プロジェクトを作成してから、エクスポートされたファイルを復元します。ただし、プロジェクトをゼロから復元するには、一部のプロジェクトが他のプロジェクトに依存していることから特定の順序を指定する必要があります。たとえば、**pods** を作成する前に **configmaps** を作成する必要があります。

手順

1. プロジェクトが単一ファイルとしてエクスポートされている場合は、以下のコマンドを実行してこのファイルをインポートします。

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

警告

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

2. **project_export.sh** スクリプトを使用してプロジェクトをエクスポートした場合、ファイルは **projectname** ディレクトリーにあります。それらは、**project_import.sh** スクリプトを再度実行してインポートできます。このスクリプトは適切な順序で **oc create** プロセスを実行します。

```
$ mkdir ~/git
$ cd ~/git
$ git clone https://github.com/openshift/openshift-ansible-
contrib.git
$ cd openshift-ansible-contrib/reference-
```

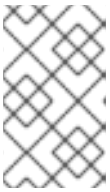
```
architecture/day2ops/scripts
$ ./project_import.sh <projectname_path>
```

例:

```
$ ls ~/backup/myproject
bcs.json           dc_guestbook_patched.json        dc_ruby-
ex_patched.json  pvcs.json          secrets.json
builds.json        dc_hello-openshift.json          iss.json
rcs.json           serviceaccounts.json
cms.json           dc_hello-openshift_patched.json  ns.json
rolebindings.json  svcs.json
dc_guestbook.json  dc_ruby-ex.json                  pods.json
routes.json        templates.json

$ ./project_import.sh ~/backup/myproject
namespace "myproject" created
rolebinding "admin" created
rolebinding "system:deployers" created
rolebinding "system:image-builders" created
rolebinding "system:image-pullers" created
secret "builder-dockercfg-mqhs6" created
secret "default-dockercfg-51xb9" created
secret "deployer-dockercfg-6kvz7" created
Error from server (AlreadyExists): error when creating
"myproject//serviceaccounts.json": serviceaccounts "builder" already
exists
Error from server (AlreadyExists): error when creating
"myproject//serviceaccounts.json": serviceaccounts "default" already
exists
Error from server (AlreadyExists): error when creating
"myproject//serviceaccounts.json": serviceaccounts "deployer"
already exists
error: no objects passed to create
service "guestbook" created
service "hello-openshift" created
service "ruby-ex" created
imagestream "guestbook" created
imagestream "hello-openshift" created
imagestream "ruby-22-centos7" created
imagestream "ruby-ex" created
error: no objects passed to create
error: no objects passed to create
buildconfig "ruby-ex" created
build "ruby-ex-1" created
deploymentconfig "guestbook" created
deploymentconfig "hello-openshift" created
deploymentconfig "ruby-ex" created
replicationcontroller "ruby-ex-1" created
Error from server (AlreadyExists): error when creating
"myproject//rcs.json": replicationcontrollers "guestbook-1" already
exists
Error from server (AlreadyExists): error when creating
"myproject//rcs.json": replicationcontrollers "hello-openshift-1"
already exists
pod "guestbook-1-c010g" created
```

```
pod "hello-openshift-1-4zw2q" created
pod "ruby-ex-1-rxc74" created
Error from server (AlreadyExists): error when creating
"myproject//pods.json": object is being deleted: pods "ruby-ex-1-
build" already exists
error: no objects passed to create
```



注記

serviceaccounts およびシークレットなどの一部のオブジェクトがプロジェクト作成時に自動的に作成されるために **AlreadyExists** エラーが表示される場合があります。

3. **buildconfigs** を使用するかどうかを確認します。

```
$ oc get bc
NAME      TYPE      FROM      LATEST
ruby-ex   Source    Git       1
$ oc get pods
NAME                      READY     STATUS    RESTARTS   AGE
guestbook-1-plnnq         1/1       Running   0          26s
hello-openshift-1-g4g0j   1/1       Running   0          26s
```

buildconfigs を使用する場合、ビルドは自動的にトリガーされず、アプリケーションは実行されません。

4. **buildconfigs** を使用する場合、ビルドをトリガーするには **oc start-build** コマンドを実行します。

```
$ for bc in $(oc get bc -o jsonpath="{.items[*].metadata.name}")
do
    oc start-build ${bc}
done
```

Pod はビルドの完了後にデプロイされます。

5. プロジェクトが復元されたことを確認するには、以下を実行します。

```
$ oc get all
NAME         TYPE      FROM      LATEST
bc/ruby-ex   Source    Git       2

NAME               TYPE      FROM          STATUS
STARTED              DURATION
builds/ruby-ex-1   Source    Git           Error (BuildPodDeleted)
About a minute ago
builds/ruby-ex-2   Source    Git@c457001   Complete
55 seconds ago       12s

NAME                 DOCKER REPO
TAGS      UPDATED
is/guestbook         10.111.255.221:5000/myproject/guestbook
latest    About a minute ago
is/hello-openshift   10.111.255.221:5000/myproject/hello-openshift
```

```
latest    About a minute ago
is/ruby-22-centos7   10.111.255.221:5000/myproject/ruby-22-centos7
latest    About a minute ago
is/ruby-ex           10.111.255.221:5000/myproject/ruby-ex
latest    43 seconds ago

NAME                 REVISION   DESIRED   CURRENT   TRIGGERED BY
dc/guestbook         1          1         1
config,image(guestbook:latest)
dc/hello-openshift   1          1         1
config,image(hello-openshift:latest)
dc/ruby-ex           1          1         1
config,image(ruby-ex:latest)

NAME                   DESIRED   CURRENT   READY     AGE
rc/guestbook-1         1         1         1         1m
rc/hello-openshift-1   1         1         1         1m
rc/ruby-ex-1           1         1         1         43s

NAME                  CLUSTER-IP       EXTERNAL-IP   PORT(S)
AGE
svc/guestbook         10.111.126.115   <none>        3000/TCP
1m
svc/hello-openshift   10.111.23.21     <none>
8080/TCP,8888/TCP   1m
svc/ruby-ex           10.111.162.157   <none>        8080/TCP
1m

NAME                          READY     STATUS      RESTARTS   AGE
po/guestbook-1-plnnq          1/1       Running     0          1m
po/hello-openshift-1-g4g0j    1/1       Running     0          1m
po/ruby-ex-1-h99np            1/1       Running     0          42s
po/ruby-ex-2-build            0/1       Completed   0          55s
```

注記

サービスおよび Pod IP アドレスは作成時に動的に割り当てられるので異なっています。

6.3. PERSISTENT VOLUME CLAIM (永続ボリューム要求) のバックアップ

コンテナー内の永続データをサーバーと同期できます。

重要

OpenShift Container Platform 環境をホストする一部のプロバイダーでは、バックアップおよび復元目的でサードパーティーのスナップショットサービスを起動する機能がある場合があります。ただし、OpenShift Container Platform ではこれらのサービスを起動する機能を提供していないため、本書ではこれらの手順については説明しません。

特定のアプリケーションに関する正しいバックアップ手順は、製品のドキュメントを参照してください。たとえば、mysql データディレクトリー自体をコピーしても使用可能なバックアップは作成されません。その代わりに、OpenShift Container Platform がホストするプラットフォームで提供されるス

ナップショットソリューションを使用するなど、関連のアプリケーション特有のバックアップ手順を実行してから、データを同期してください。

手順

1. プロジェクトおよび Pod を表示します。

```
$ oc get pods
NAME           READY     STATUS      RESTARTS   AGE
demo-1-build   0/1       Completed   0          2h
demo-2-fxx6d   1/1       Running     0          1h
```

2. 永続ボリュームで使用されているボリュームを検索できるように必要な Pod の情報を取得します。

```
$ oc describe pod demo-2-fxx6d
Name:   demo-2-fxx6d
Namespace:  test
Security Policy: restricted
Node:   ip-10-20-6-20.ec2.internal/10.20.6.20
Start Time:  Tue, 05 Dec 2017 12:54:34 -0500
Labels:   app=demo
   deployment=demo-2
   deploymentconfig=demo
Status:   Running
IP:   172.16.12.5
Controllers:  ReplicationController/demo-2
Containers:
  demo:
    Container ID:
docker://201f3e55b373641eb36945d723e1e212ecab847311109b5cee1fd010942
4217a
    Image:  docker-
registry.default.svc:5000/test/demo@sha256:0a9f2487a0d95d51511e49d20
dc9ff6f350436f935968b0c83fcb98a7a8c381a
    Image ID:  docker-pullable://docker-
registry.default.svc:5000/test/demo@sha256:0a9f2487a0d95d51511e49d20
dc9ff6f350436f935968b0c83fcb98a7a8c381a
    Port:  8080/TCP
    State:  Running
      Started:  Tue, 05 Dec 2017 12:54:52 -0500
    Ready:  True
    Restart Count: 0
    Volume Mounts:
      */opt/app-root/src/uploaded from persistent-volume (rw)*
      /var/run/secrets/kubernetes.io/serviceaccount from default-
token-8mmrk (ro)
    Environment Variables: <none>
...omitted...
```

この出力は永続データが **/opt/app-root/src/uploaded** ディレクトリーにあることを示しています。

3. データをローカルにコピーします。

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

```
receiving incremental file list
uploaded/
uploaded/ocp_sop.txt
uploaded/lost+found/

sent 38 bytes  received 190 bytes  152.00 bytes/sec
total size is 32  speedup is 0.14
```

ocp_sop.txt ファイルはローカルシステムにダウンロードされ、バックアップソフトウェアまたは別のバックアップメカニズムでバックアップされます。

注記

また、**pvc** を使用せずに Pod を起動する場合には、直前の手順を使用することもできますが、**pvc** が必要かどうかを後で確認する必要があります。データを保存してから復元プロセスを使用し、新規ストレージを生成することができます。

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

バックアップした Persistent Volume Claim (永続ボリューム要求、PVC) データを復元することができます。ファイルを削除してからそのファイルを予想される場所に戻すか、または Persistent Volume Claim (永続ボリューム要求) を移行することができます。ストレージを移行する必要がある場合や、バックエンドストレージがすでに存在しないなどの障害発生時には移行する必要がある場合があります。

特定のアプリケーションの適切な復元手順については、それぞれの製品ドキュメントを参照してください。

6.4.1. ファイルの既存 PVC への復元

手順

1. ファイルを削除します。

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

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

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

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

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

6.4.2. データの新規 PVC への復元

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

手順

1. 現在定義されている **claim-name** を上書きします。

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

2. Pod が新規 PVC を使用していることを確認します。

```
$ oc describe dc/demo
Name:  demo
Namespace: test
Created: 3 hours ago
Labels:  app=demo
Annotations: openshift.io/generated-by=OpenShiftNewApp
Latest Version: 3
Selector: app=demo,deploymentconfig=demo
Replicas: 1
Triggers: Config, Image(demo@latest, auto=true)
Strategy: Rolling
Template:
  Labels: app=demo
  deploymentconfig=demo
  Annotations: openshift.io/container.demo.image.entrypoint=
["container-entrypoint","/bin/sh","-c","$STI_SCRIPTS_PATH/usage"]
  openshift.io/generated-by=OpenShiftNewApp
  Containers:
   demo:
    Image: docker-
registry.default.svc:5000/test/demo@sha256:0a9f2487a0d95d51511e49d20
dc9ff6f350436f935968b0c83fcb98a7a8c381a
    Port: 8080/TCP
    Volume Mounts:
      /opt/app-root/src/uploaded from persistent-volume (rw)
    Environment Variables: <none>
  Volumes:
   persistent-volume:
    Type: PersistentVolumeClaim (a reference to a
PersistentVolumeClaim in the same namespace)
    *ClaimName: filestore*
    ReadOnly: false
...omitted...
```

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

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

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

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

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

6.5. イメージおよびコンテナーのプルーニング

収集されたデータおよびオブジェクトの古いバージョンのプルーニングについての詳細は、「Pruning Resources (リソースのプルーニング)」のトピックを参照してください。

第7章 DOCKER タスク

OpenShift Container Platform は Docker を使用して、任意の数のコンテナーで作成される Pod でアプリケーションを実行します。

クラスター管理者は、OpenShift Container Platform インストールの各種の要素を効率的に実行するために Docker に追加の設定を加える必要がある場合があります。

7.1. DOCKER ストレージの拡張

利用可能なストレージの容量を増やすことにより、停止が発生しない継続的なデプロイメントが可能になります。これを実行するには、適切なサイズの空き容量を含む空きのパーティションが利用可能でなければなりません。

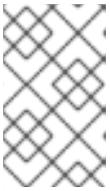
7.1.1. ノードの退避

手順

1. マスターインスタンスからか、またはクラスター管理者として、ノードからの Pod の退避を許可し、そのノードでの他の Pod のスケジューリングを無効にします。

```
$ NODE=ose-app-node01.example.com
$ oc adm manage-node ${NODE} --schedulable=false
NAME                           STATUS                     AGE
VERSION
ose-app-node01.example.com   Ready,SchedulingDisabled   20m
v1.6.1+5115d708d7

$ oc adm drain ${NODE} --ignore-daemonsets
node "ose-app-node01.example.com" already cordoned
pod "perl-1-build" evicted
pod "perl-1-3lnsh" evicted
pod "perl-1-9jzd8" evicted
node "ose-app-node01.example.com" drained
```



注記

移行されないローカルボリュームでコンテナーが実行されている場合には、以下のコマンドを実行します。**`oc adm drain ${NODE} --ignore-daemonsets --delete-local-data`**.

2. ノード上の Pod を一覧表示し、それらが削除されていることを確認します。

```
$ oc adm manage-node ${NODE} --list-pods

Listing matched pods on node: ose-app-node01.example.com

NAME       READY     STATUS     RESTARTS    AGE
```

3. ノードごとに、これまでの 2 つの手順を繰り返します。

Pod またはノードの退避またはドレイン (解放) についての詳細は、「ノードの保守」を参照してください。

7.1.2. ストレージの拡張

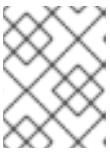
Docker ストレージは、新規ディスクを割り当てるか、または既存ディスクを拡張するかの 2 つの方法で拡張できます。

新規ディスクによるストレージの拡張

前提条件

- 新規ディスクは、追加のストレージを必要とする既存のインスタンスで利用可能でなければなりません。以下の手順では、**/etc/sysconfig/docker-storage-setup** ファイルに示されているように、元のディスクに **/dev/xvdb** というラベルが付けられ、新規ディスクには **/dev/xvdd** というラベルが付けられています。

```
# vi /etc/sysconfig/docker-storage-setup
DEVS="/dev/xvdb /dev/xvdd"
```



注記

プロセスは基礎となる OpenShift Container Platform インフラストラクチャーによって異なります。

手順

1. **docker** および **atomic-openshift-node** サービスを停止します。

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

2. **docker-storage-setup** コマンドを実行してコンテナーストレージに関連付けられたボリュームグループおよび論理ボリュームを拡張します。

```
# docker-storage-setup
INFO: Volume group backing root filesystem could not be determined
INFO: Device /dev/xvdb is already partitioned and is part of volume
group docker_vol
INFO: Device node /dev/xvdd1 exists.
  Physical volume "/dev/xvdd1" successfully created.
  Volume group "docker_vol" successfully extended
```

3. Docker サービスを起動します。

```
# systemctl start docker
# vgs
  VG         #PV #LV #SN Attr   VSize  VFree
  docker_vol   2   1   0 wz--n- 64.99g <55.00g
```

4. 新規ボリュームグループを作成して **docker-storage-setup** を再実行する場合と比較したディスクの追加のメリットとして、システムで使用されていたイメージが新規ストレージの追加後もそのまま存在するという点があります。

```
# docker images
REPOSITORY                                                  TAG
IMAGE ID            CREATED             SIZE
docker-registry.default.svc:5000/tet/perl                   latest
```

```
8b0b0106fb5e        13 minutes ago      627.4 MB
registry.access.redhat.com/rhscl/perl-524-rhel7         <none>
912b01ac7570        6 days ago          559.5 MB
registry.access.redhat.com/openshift3/ose-deployer
v3.6.173.0.21       89fd398a337d        5 weeks ago         970.2
MB
registry.access.redhat.com/openshift3/ose-pod
v3.6.173.0.21       63accd48a0d7        5 weeks ago         208.6
MB
```

5. ストレージ容量を拡張した状態で、新しく入ってくる Pod を受け入れられるようにノードをスケジュール可能な状態にします。
クラスター管理者として、マスターインスタンスから以下を実行します。

```
$ oc adm manage-node ${NODE} --schedulable=true

ose-master01.example.com   Ready,SchedulingDisabled   24m
v1.6.1+5115d708d7
ose-master02.example.com   Ready,SchedulingDisabled   24m
v1.6.1+5115d708d7
ose-master03.example.com   Ready,SchedulingDisabled   24m
v1.6.1+5115d708d7
ose-infra-node01.example.com   Ready                     24m
v1.6.1+5115d708d7
ose-infra-node02.example.com   Ready                     24m
v1.6.1+5115d708d7
ose-infra-node03.example.com   Ready                     24m
v1.6.1+5115d708d7
ose-app-node01.example.com   Ready                     24m
v1.6.1+5115d708d7
ose-app-node02.example.com   Ready                     24m
v1.6.1+5115d708d7
```

新規ディスクによるストレージの拡張

1. 直前の手順に従って、ノードを退避します。
2. **docker** および **atomic-openshift-node** サービスを停止します。

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

3. 既存ディスクを必要に応じてサイズ変更します。これは環境に応じて異なります。
   - LVM (Logical Volume Manager) を使用している場合:
     - 論理ボリュームの削除:

       ```
       # lvremove /dev/docker_vg/docker/lv
       ```

     - Docker ボリュームグループの削除:

       ```
       # vgremove docker_vg
       ```

     - 物理ボリュームの削除:

```
# pvremove /dev/<my_previous_disk_device>
```

- クラウドプロバイダーを使用している場合は、ディスクの割り当てを解除し、ディスクを破棄してから新規のより大きなディスクを作成し、これをインスタンスに割り当てることができます。
- クラウド以外の環境の場合、ディスクおよびファイルシステムのサイズは変更することができます。詳細については、以下のソリューションを参照してください。
  - https://access.redhat.com/solutions/199573

4. デバイス名、サイズなどを確認して **/etc/sysconfig/container-storage-setup** ファイルが新規ディスクに対して適切に設定されていることを確認します。
5. **docker-storage-setup** を実行して新規ディスクを再設定します。

```
# docker-storage-setup
INFO: Volume group backing root filesystem could not be determined
INFO: Device /dev/xvdb is already partitioned and is part of volume
group docker_vol
INFO: Device node /dev/xvdd1 exists.
  Physical volume "/dev/xvdd1" successfully created.
  Volume group "docker_vol" successfully extended
```

6. Docker サービスを起動します。

```
# systemctl start docker
# vgs
  VG         #PV #LV #SN Attr   VSize  VFree
  docker_vol   2   1   0 wz--n- 64.99g <55.00g
```

7. **atomic-openshift-node** サービスを起動します。

```
# systemctl start atomic-openshift-node
```

7.1.3. ストレージバックエンドの変更

サービスおよびファイルシステムの機能拡張に応じて、新規機能を使用できるようにストレージバックエンドの変更が必要になる場合があります。以下の手順では、デバイスマッパーのバックエンドを **overlay2** ストレージのバックエンドに変更する例について示しています。**overlay2** は従来のデバイスマッパーに対して高いスピードと密度を提供します。

7.1.3.1. ノードの退避

1. マスターインスタンスからか、またはクラスター管理者として、ノードからの Pod の退避を許可し、そのノードでの他の Pod のスケジューリングを無効にします。

```
$ NODE=ose-app-node01.example.com
$ oc adm manage-node ${NODE} --schedulable=false
NAME                          STATUS                     AGE
VERSION
ose-app-node01.example.com    Ready,SchedulingDisabled   20m
v1.6.1+5115d708d7
```

```
$ oc adm drain ${NODE} --ignore-daemonsets
node "ose-app-node01.example.com" already cordoned
pod "perl-1-build" evicted
pod "perl-1-3lnsh" evicted
pod "perl-1-9jzd8" evicted
node "ose-app-node01.example.com" drained
```

注記

移行されないローカルボリュームでコンテナーが実行されている場合には、以下のコマンドを実行します。**oc adm drain ${NODE} --ignore-daemonsets --delete-local-data**

2. ノード上の Pod を一覧表示し、それらが削除されていることを確認します。

```
$ oc adm manage-node ${NODE} --list-pods

Listing matched pods on node: ose-app-node01.example.com

NAME      READY     STATUS    RESTARTS   AGE
```

Pod またはノードの退避またはドレイン (解放) についての詳細は、「ノードの保守」を参照してください。

3. コンテナーが現時点でインスタンス上で実行されていない状態で、**docker** および **atomic-openshift-node service** サービスを停止します。

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

4. ボリュームグループの名前、論理グループ名、および物理ボリューム名を確認します。

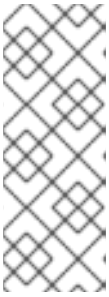
```
# vgs
  VG         #PV #LV #SN Attr   VSize   VFree
  docker_vol   1   1   0 wz--n- <25.00g 15.00g

# lvs
LV       VG         Attr       LSize   Pool Origin Data%  Meta%
Move Log Cpy%Sync Convert
 dockerlv docker_vol -wi-ao---- <10.00g

# lvremove /dev/docker_vol/docker-pool  -y
# vgremove docker_vol -y
# pvs
  PV         VG         Fmt  Attr PSize   PFree
  /dev/xvdb1 docker_vol lvm2 a--  <25.00g 15.00g

# pvremove /dev/xvdb1 -y
# rm -Rf /var/lib/docker/*
# rm -f /etc/sysconfig/docker-storage
```

5. **docker-storage-setup** ファイルを **STORAGE_DRIVER** を指定するように変更します。



注記

システムが Red Hat Enterprise Linux バージョン 7.3 から 7.4 にアップグレードする際に、**docker** サービスは **/var** を extfs の **STORAGE_DRIVER** と共に使用しようとします。ただし、extfs を **STORAGE_DRIVER** として使用するとエラーが発生します。このエラーについての詳細は、以下のバグを参照してください。

- Bugzilla ID: 1490910

```
DEVS=/dev/xvdb
VG=docker_vol
DATA_SIZE=95%VG
STORAGE_DRIVER=overlay2
CONTAINER_ROOT_LV_NAME=dockerlv
CONTAINER_ROOT_LV_MOUNT_PATH=/var/lib/docker
CONTAINER_ROOT_LV_SIZE=100%FREE
```

6. ストレージをセットアップします。

```
# docker-storage-setup
```

7. **docker** および **atomic-openshift-node** サービスを起動します。

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

8. 使用するストレージを **overlay2** に変更して、新しく入ってくる Pod を受け入れられるようにノードをスケジュール可能な状態にします。
マスターインスタンスから、またはクラスター管理者として以下を実行します。

```
$ oc adm manage-node ${NODE} --schedulable=true

ose-master01.example.com   Ready,SchedulingDisabled   24m
v1.6.1+5115d708d7
ose-master02.example.com   Ready,SchedulingDisabled   24m
v1.6.1+5115d708d7
ose-master03.example.com   Ready,SchedulingDisabled   24m
v1.6.1+5115d708d7
ose-infra-node01.example.com   Ready                      24m
v1.6.1+5115d708d7
ose-infra-node02.example.com   Ready                      24m
v1.6.1+5115d708d7
ose-infra-node03.example.com   Ready                      24m
v1.6.1+5115d708d7
ose-app-node01.example.com   Ready                     24m
v1.6.1+5115d708d7
ose-app-node02.example.com   Ready                     24m
v1.6.1+5115d708d7
```

7.2. DOCKER 証明書の管理

OpenShift Container Platform 内部レジストリーは Pod として作成されます。ただし、コンテナーは必要な場合は外部レジストリーからプルされます。デフォルトでは、レジストリーは TCP ポート 5000 をリッスンします。レジストリーは TLS 経由でエクスポートされたイメージのセキュリティーを保護す

るか、またはトラフィックを暗号化せずにレジストリーを実行するオプションを提供します。

警告

Docker は **.crt** ファイルを CA 証明書として、**.cert** ファイルをクライアント証明書として解釈します。CA 拡張は **.crt である必要があります**。

7.2.1. 外部レジストリー用の認証局証明書のインストール

OpenShift Container Platform を外部レジストリーで使用するために、レジストリーの認証局 (CA) 証明書が、レジストリーからイメージをプルできるすべてのノードについて信頼されている必要があります。

注記

Docker のバージョンによっては、Docker レジストリーを信頼するプロセスは異なります。最新バージョンの Docker のルート認証局はシステムのデフォルトにマージされています。**docker** バージョン 1.13 よりも前のバージョンでは、システムのデフォルト証明書は他のカスタムルート証明書が存在しない場合にのみ使用されます。

手順

1. CA 証明書を **/etc/pki/ca-trust/source/anchors/** にコピーします。

```
$ sudo cp myregistry.example.com.crt /etc/pki/ca-
trust/source/anchors/
```

2. CA 証明書を展開し、信頼される認証局の一覧に追加します。

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

3. **openssl** コマンドを使用して SSL 証明書を確認します。

```
$ openssl verify myregistry.example.com.crt
myregistry.example.com.crt: OK
```

4. 証明書が配置され、信頼が更新されたら、**docker** サービスを再起動して新規証明書が適切に設定されていることを確認します。

```
$ sudo systemctl restart docker.service
```

Docker バージョンの 1.13 よりも前のバージョンの場合は、認証局を信頼するために以下の追加の手順を実行します。

1. すべてのノードで、ディレクトリーの名前が Docker レジストリーのホスト名となっている新規ディレクトリーを **/etc/docker/certs.d** に作成します。

```
$ sudo mkdir -p /etc/docker/certs.d/myregistry.example.com
```

注記

ポート番号は、Docker レジストリーがポート番号なしにアクセスできない場合を除き不要です。ポートを元の Docker レジストリーにポイントするには、以下を使用します。**myregistry.example.com:port**

2. Docker レジストリーに IP アドレス経由でアクセスするには、ディレクトリーの名前が Docker レジストリーの IP である新規ディレクトリーを、すべてのノードの **/etc/docker/certs.d** 内に作成する必要があります。

```
$ sudo mkdir -p /etc/docker/certs.d/10.10.10.10
```

3. CA 証明書を直前の手順で新たに作成された Docker ディレクトリーにコピーします。

```
$ sudo cp myregistry.example.com.crt \
  /etc/docker/certs.d/myregistry.example.com/ca.crt

$ sudo cp myregistry.example.com.crt
/etc/docker/certs.d/10.10.10.10/ca.crt
```

4. 証明書がコピーされた後に、**docker** サービスを再起動して新規証明書が使用されていることを確認します。

```
$ sudo systemctl restart docker.service
```

7.2.2. Docker 証明書のバックアップ

ノードホストのバックアップの実行時に、外部レジストリーの証明書を組み込みます。

手順

1. **/etc/docker/certs.d** を使用している場合、ディレクトリーに含まれるすべての証明書をコピーし、ファイルを保存します。

```
$ sudo tar -czvf docker-registry-certs-$(hostname)-$(date
+%Y%m%d).tar.gz /etc/docker/certs.d/
```

2. システムの信頼を使用する場合、証明書をシステムの信頼内に追加する前に保存します。保存が完了したら、**trust** コマンドを使用して復元するために証明書を展開します。システム信頼の CA を特定し、**pkcs11** ID を書き留めておきます。

```
$ trust list
...[OUTPUT OMMITED]...
pkcs11:id=%a5%b3%e1%2b%2b%49%b6%d7%73%a1%aa%94%f5%01%e7%73%65%4c%ac%
50;type=cert
    type: certificate
    label: MyCA
    trust: anchor
    category: authority
...[OUTPUT OMMITED]...
```

3. **pem** 形式の証明書を展開し、これに名前を指定します (例: **myca.crt**)。

```
$ trust extract --format=pem-bundle \
 --
filter="%a5%b3%e1%2b%2b%49%b6%d7%73%a1%aa%94%f5%01%e7%73%65%4c%ac%50
;type=cert" myca.crt
```

4. 証明書が **openssl** で適切に展開されていることを確認します。

```
$ openssl verify myca.crt
```

5. 必要なすべての証明書についてこの手順を繰り返し、ファイルをリモートの場所に保存します。

7.2.3. Docker 証明書の復元

外部レジストリー用の Docker 証明書が削除されるか、または破損している場合、復元メカニズムでは先に実行されたバックアップのファイルを使用して、インストール方法と同じ手順を使用します。

7.3. DOCKER レジストリーの管理

OpenShift Container Platform を、外部 **docker** レジストリーを使用してイメージをプルできるように設定できます。ただし、設定ファイルを使用して特定のイメージまたはレジストリーを許可したり、拒否したりすることもできます。

外部レジストリーがネットワークトラフィックの証明書を使用して公開される場合、これにはセキュアなレジストリーとしての名前を付けることができます。そうしない場合には、レジストリーとホスト間のトラフィックはプレーンなテキストで行われ、暗号化されないため、これが非セキュアなレジストリーになります。

7.3.1. Docker search の外部レジストリー

デフォルトで、**docker** デーモンにはイメージを任意のレジストリーからプルできる機能がありますが、検索操作は **docker.io/** および **registry.access.redhat.com** に対して実行されます。デーモンは、**--add-registry** オプションを **docker** デーモンに指定してイメージを他のレジストリーから検索できるように設定できます。

注記

Red Hat Registry **registry.access.redhat.com** からイメージを検索する機能はデフォルトで Red Hat Enterprise Linux **docker** パッケージにあります。

手順

1. ユーザーが **docker search** に他のレジストリーを指定してイメージを検索できるようにするには、それらのレジストリーを **registries** パラメーターの下の **/etc/containers/registries.conf** ファイルに追加します。

```
registries:
  - registry.access.redhat.com
  - my.registry.example.com
```

OpenShift Container Platform バージョン 3.6 よりも前のバージョンでは、これは **/etc/sysconfig/docker** に以下のオプションを指定して実行されました。

```
ADD_REGISTRY="--add-registry=registry.access.redhat.com --add-
registry=my.registry.example.com"
```

最初に追加されるレジストリーが最初に検索されるレジストリーになります。

2. **docker** デーモンを再起動し、**my.registry.example.com** の使用を許可します。

```
$ sudo systemctl restart docker.service
```

docker デーモンの再起動により、**docker** コンテナーが再起動します。

3. Ansible インストーラーを使用する場合、これは Ansible ホストファイルの **openshift_docker_additional_registries** 変数を使用して設定できます。

```
openshift_docker_additional_registries=registry.access.redhat.com,my
.registry.example.com
```

7.3.2. Docker 外部レジストリーのホワイトリストおよびブラックリスト

Docker は、**registries** および **block_registries** フラグを **docker** デーモンに設定して、外部レジストリーからの操作をブロックするように設定できます。

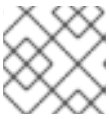
手順

1. 許可されるレジストリーを **registries** フラグの付いた**/etc/containers/registries.conf** ファイルに追加します。

```
registries:
  - registry.access.redhat.com
  - my.registry.example.com
```

3.6 よりも前のバージョンでは、**/etc/sysconfig/docker** ファイルが代わりに変更されます。

```
ADD_REGISTRY="--add-registry=registry.access.redhat.com --add-
registry=my.registry.example.com"
```



注記

docker.io レジストリーは同じ方法で追加できます。

2. 残りのレジストリーをブロックします。

```
block_registries:
   - all
```

3. 古いバージョンの残りのレジストリーをブロックします。

```
BLOCK_REGISTRY='--block-registry=all'
```

4. **docker** デーモンを再起動します。

```
$ sudo systemctl restart docker.service
```

docker デーモンの再起動により、**docker** コンテナーが再起動します。

5. この例では、**docker.io** レジストリーがブラックリストに入れられているため、そのレジストリーに関連するすべての操作が失敗します。

```
$ sudo docker pull hello-world
Using default tag: latest
Trying to pull repository registry.access.redhat.com/hello-world ...
Trying to pull repository my.registry.example.com/hello-world ...
Trying to pull repository registry.access.redhat.com/hello-world ...
unknown: Not Found
$ sudo docker pull docker.io/hello-world
Using default tag: latest
Trying to pull repository docker.io/library/hello-world ...
All endpoints blocked.
```

ファイルを再び変更し、サービスを再起動して **docker.io** を **registries** 変数に追加します。

```
registries:
  - registry.access.redhat.com
  - my.registry.example.com
  - docker.io
block_registries:
  - all
```

または、以下のようにします。

```
ADD_REGISTRY="--add-registry=registry.access.redhat.com --add-
registry=my.registry.example.com --add-registry=docker.io"
BLOCK_REGISTRY='--block-registry=all'
```

6. Docker サービスを再起動します。

```
$ sudo systemctl restart docker
```

7. イメージがプルできる状態であることを確認するには、以下を実行します。

```
$ sudo docker pull docker.io/hello-world
Using default tag: latest
Trying to pull repository docker.io/library/hello-world ...
latest: Pulling from docker.io/library/hello-world

9a0669468bf7: Pull complete
Digest:
sha256:0e06ef5e1945a718b02a8c319e15bae44f47039005530bc617a5d071190ed
3fc
```

8. 外部レジストリーを使用する必要がある場合 (レジストリーを使用する必要のあるすべてのノードホストの **docker** デーモン設定ファイルを変更する場合など)、これらのノードでブラックリストを作成し、悪意あるコンテナーが実行されるのを防ぐことができます。

Ansible インストーラーを使用する場合、これは、Ansible ホストファイルの **openshift_docker_additional_registries** および **openshift_docker_blocked_registries** 変数を使用して設定できます。

```
openshift_docker_additional_registries=registry.access.redhat.com,my
.registry.example.com
openshift_docker_blocked_registries=all
```

7.3.3. セキュアなレジストリー

イメージを外部レジストリーからプルできるようにするには、レジストリー証明書を信頼できる必要があります。そうでない場合には、イメージのプル操作は失敗します。

これを実行する場合、「外部レジストリーの認証局証明書のインストール」のセクションを参照してください。

ホワイトリストを使用する場合、外部レジストリーは上記のように**registries** 変数に追加されるはずです。

7.3.4. 非セキュアなレジストリー

信頼されていない証明書を使用する外部レジストリー、または証明書がまったく使用されない外部レジストリーは使用しないでください。

ただし、非セキュアなレジストリーは、**docker** デーモンがリポジトリーからイメージをプルできるように **--insecure-registry** オプションを使用して追加する必要があります。これは **--add-registry** オプションと同じですが、**docker** 操作については検証されていません。

レジストリーはこれらの両方のオプションを使用して追加される必要があります。これにより、検索が可能になり、ブラックリストがある場合はイメージのプルなどの他の操作を実行することができます。

テストとして、以下に **localhost** の非セキュアなレジストリーを追加する方法についての例を示します。

手順

1. **/etc/containers/registries.conf** 設定ファイルを localhost の非セキュアなレジストリーを追加するように変更します。

```
registries:
  - registry.access.redhat.com
  - my.registry.example.com
  - docker.io
insecure_registries:
  - localhost:5000
block_registries:
  - all
```

3.6 よりも前のバージョンの場合、**/etc/sysconfig/docker** 設定ファイルを変更して localhost を追加します。

```
ADD_REGISTRY="--add-registry=registry.access.redhat.com --add-
registry=my.registry.example.com --add-registry=docker.io --add-
registry=localhost:5000"
```

```
INSECURE_REGISTRY="--insecure-registry=localhost:5000"
BLOCK_REGISTRY='--block-registry=all'
```

2. **docker** デーモンを再起動してレジストリーを使用します。

```
$ sudo systemctl restart docker.service
```

docker デーモンを再起動することにより、**docker** コンテナーが再起動されます。

3. Docker レジストリー Pod を **localhost** で実行します。

```
$ sudo docker run -p 5000:5000 registry:2
```

4. イメージをプルします。

```
$ sudo docker pull openshift/hello-openshift
```

5. イメージにタグを付けます。

```
$ sudo docker tag docker.io/openshift/hello-openshift:latest
localhost:5000/hello-openshift-local:latest
```

6. イメージをローカルレジストリーにプッシュします。

```
$ sudo docker push localhost:5000/hello-openshift-local:latest
```

7. Ansible インストーラーを使用する場合、これは、**Ansible** ホストファイルの **openshift_docker_additional_registries**、**openshift_docker_blocked_registries**、および **openshift_docker_insecure_registries** 変数を使用して設定できます。

```
openshift_docker_additional_registries=registry.access.redhat.com,my
.registry.example.com,localhost:5000
openshift_docker_insecure_registries=localhost:5000
openshift_docker_blocked_registries=all
```

注記

ホストの IP アドレスに **openshift_docker_insecure_registries** 変数も設定できます。**0.0.0.0/0** は有効な設定ではありません。

7.3.5. 認証済みレジストリー

認証済みのレジストリーを **docker** で使用するには、**docker** デーモンがレジストリーにログインする必要があります。OpenShift Container Platform の場合、ユーザーはホストで **docker login** コマンドを実行できないため、異なる手順セットを実行する必要があります。認証済みのレジストリーはユーザーがプルできるイメージや外部レジストリーにアクセスできるユーザーを制限するために使用できます。

外部 **docker** レジストリーに認証が必要な場合は、レジストリーを使用するプロジェクトで特殊なシークレットを作成してから、そのシークレットを使用して **docker** 操作を実行します。

手順

1. ユーザーが **docker** レジストリーにログインするプロジェクトで **dockercfg** シークレットを作成します。

```
$ oc project <my_project>
$ oc create secret docker-registry <my_registry> --docker-server=
<my.registry.example.com> --docker-username=<username> --docker-
password=<my_password> --docker-email=<me@example.com>
```

2. **.dockercfg** ファイルが存在する場合、**oc** コマンドを使用してシークレットを作成します。

```
$ oc create secret generic <my_registry> --from-file=.dockercfg=
<path/to/.dockercfg> --type=kubernetes.io/dockercfg
```

3. **$HOME/.docker/config.json** ファイルを設定します。

```
$ oc create secret generic <my_registry> --from-
file=.dockerconfigjson=<path/to/.dockercfg> --
type=kubernetes.io/dockerconfigjson
```

4. **dockercfg** を使用し、シークレットをプル操作を実行するサービスアカウントにリンクして、イメージを認証済みレジストリーからプルします。イメージをプルするためのデフォルトのサービスアカウント名は **default** です。

```
$ oc secrets link default <my_registry> --for=pull
```

5. S2I 機能を使用してイメージをプッシュする場合、**dockercfg** シークレットは S2I Pod にマウントされるため、これをビルドを実行する適切なサービスアカウントにリンクする必要があります。イメージをビルドするために使用されるデフォルトのサービスアカウントの名前は **builder** です。

```
$ oc secrets link builder <my_registry>
```

6. **buildconfig** では、シークレットをプッシュまたはプル操作用に指定する必要があります。

```
"type": "Source",
"sourceStrategy": {
    "from": {
        "kind": "DockerImage",
        "name": "*my.registry.example.com*/myproject/myimage:stable"
    },
    "pullSecret": {
        "name": "*mydockerregistry*"
    },
...[OUTPUT ABBREVIATED]...
"output": {
    "to": {
        "kind": "DockerImage",
        "name": "*my.registry.example.com*/myproject/myimage:latest"
    },
    "pushSecret": {
        "name": "*mydockerregistry*"
    },
...[OUTPUT ABBREVIATED]...
```

7. 外部レジストリーが認証を外部サービスに委任する場合は、レジストリー URL を使用するレジストリー用のシークレットと、独自の URL を使用する外部の認証システム用の両方の **dockercfg** シークレットを作成します。これら両方のシークレットをサービスアカウントに追加する必要があります。

```
$ oc project <my_project>
$ oc create secret docker-registry <my_registry> --docker-server=*
<my_registry_example.com> --docker-username=<username> --docker-
password=<my_password> --docker-email=<me@example.com>
$ oc create secret docker-registry <my_docker_registry_ext_auth> --
docker-server=<my.authsystem.example.com> --docker-username=
<username> --docker-password=<my_password> --docker-email=
<me@example.com>
$ oc secrets link default <my_registry> --for=pull
$ oc secrets link default <my_docker_registry_ext_auth> --for=pull
$ oc secrets link builder <my_registry>
$ oc secrets link builder <my_docker_registry_ext_auth>
```

7.3.6. ImagePolicy 受付プラグイン

受付制御プラグインは API への要求をインターセプトし、設定されているルールに応じてチェックを実行し、それらのルールに基づいて特定のアクションを許可したり、拒否したりします。OpenShift Container Platform は、「**ImagePolicy** 受付プラグイン」を使用して、この環境で実行されている、許可済みのイメージを制限できます。以下を制御できます。

- イメージのソース: イメージのプルに使用できるレジストリー
- 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、イミュータブルなダイジェストによる Pod の実行を強制する。
- コンテナーイメージラベルの制限: イメージに特定のラベルを持たせるか、または持たせないよう強制する。
- イメージアノテーションの制限: 統合コンテナーレジストリーのイメージに特定のアノテーションを持たせるか、または持たせないよう強制する。

警告

現時点で、**ImagePolicy** 受付プラグインはベータ版とみなされています。

手順

1. **ImagePolicy** プラグインが有効にされている場合、これを、すべてのマスターノードで **/etc/origin/master/master-config.yaml** ファイルを変更して使用される外部レジストリーを許可するように変更する必要があります。

```
admissionConfig:
  pluginConfig:
    openshift.io/ImagePolicy:
      configuration:
```

```
        kind: ImagePolicyConfig
        apiVersion: v1
        executionRules:
        - name: allow-images-from-other-registries
          onResources:
          - resource: pods
          - resource: builds
          matchRegistries:
          - docker.io
          - <my.registry.example.com>
          - registry.access.redhat.com
```

注記

ImagePolicy を有効にすることにより、ユーザーはアプリケーションを使用する際にレジストリーを指定する必要があります (**oc new-app kubernetes/guestbook** ではなく **oc new-app docker.io/kubernetes/guestbook** とするなど)。そうしないと、失敗が生じます。

2. インストール時に受付プラグインを有効にするには、**openshift_master_admission_plugin_config** 変数を、すべての **pluginConfig** 設定を含む **json** でフォーマットされた文字列と共に使用できます。

```
openshift_master_admission_plugin_config=
{"openshift.io/ImagePolicy":{"configuration":
{"kind":"ImagePolicyConfig","apiVersion":"v1","executionRules":
[{"name":"allow-images-from-other-registries","onResources":
[{"resource":"pods"},{"resource":"builds"}],"matchRegistries":
["docker.io","*my.registry.example.com*","registry.access.redhat.com
"]}]}}}
```

警告

現時点で、OpenShift Container Platform 3.6.1 で修正される必要のある 1 つの問題があります。これは、**ImagePolicy** Pod はデフォルトテンプレートを使用してデプロイできず、以下のエラーメッセージを出す問題です。**Failed create | Error creating: Pod "" is invalid: spec.containers[0].\image: Forbidden: this image is prohibited by policy**

この回避策については、「Image Policy is not working as expected」という Red Hat ナレッジベースアーティクルを参照してください。

7.3.7. イメージの外部レジストリーからのインポート

アプリケーション開発者は **oc import-image** コマンドでイメージをインポートして **imagestreams** を作成でき、OpenShift Container Platform は外部レジストリーからのイメージインポートを許可または拒否するように設定できます。

手順

1. ユーザーがイメージをインポートできる許可されたレジストリーを設定するには、以下を **/etc/origin/master/master-config.yaml** ファイルに追加します。

```
imagePolicyConfig:
  allowedRegistriesForImport:
  - domainName: docker.io
  - domainName: '\*.docker.io'
  - domainName: '*.redhat.com'
  - domainName: 'my.registry.example.com'
```

2. イメージを外部認証レジストリーからインポートするには、必要なプロジェクト内にシークレットを作成します。

3. 推奨されていない場合でも、外部の認証済みレジストリーが非セキュアであるか、または証明書が信頼できない場合には、**oc import-image** コマンドを **--insecure=true** オプションを指定して使用できます。
外部の認証済みレジストリーがセキュアな場合、レジストリー証明書は、以下のようにレジストリーのインポートコントローラーを実行する際にマスターホストで信頼される必要があります。

   /etc/pki/ca-trust/source/anchors/ の証明書をコピーします。

```
$ sudo cp <my.registry.example.com.crt> /etc/pki/ca-
trust/source/anchors/<my.registry.example.com.crt>
```

4. **update-ca-trust** コマンドを実行します。

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

5. すべてのマスターホストでマスターサービスを再起動します。

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

6. 外部レジストリーの証明書は OpenShift Container Platform レジストリーで信頼されます。

```
$ for i in pem openssl java; do
  oc create configmap ca-trust-extracted-${i} --from-file
/etc/pki/ca-trust/extracted/${i}
  oc set volume dc/docker-registry --add -m /etc/pki/ca-
trust/extracted/${i} --configmap-name=ca-trust-extracted-${i} --name
ca-trust-extracted-${i}
done
```

警告

現時点で、証明書をレジストリー Pod に追加するための正式な手順はありませんが、上記の回避策を使用できます。

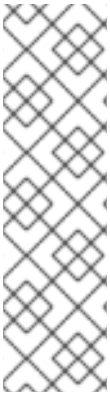
この回避策では、これらのコマンドを実行するシステムですべての信頼される証明書を使って **configmaps** を作成するため、必要な証明書のみが信頼されるクリーンなシステムからこれを実行することが推奨されます。

7. または、以下のように **Dockerfile** を使用して、イメージを再ビルドするために適切な証明書を信頼できるようレジストリーイメージを変更します。

```
FROM registry.access.redhat.com/openshift3/ose-docker-registry:v3.6
ADD <my.registry.example.com.crt> /etc/pki/ca-trust/source/anchors/
USER 0
RUN update-ca-trust extract
USER 1001
```

8. イメージを再ビルドし、これを **docker** レジストリーにプッシュし、そのイメージをレジストリー **deploymentconfig** の **spec.template.spec.containers["name":"registry"].image** として使用します。

```
$ oc patch dc docker-registry -p '{"spec":{"template":{"spec":
{"containers":
[{"name":"registry","image":"*myregistry.example.com/openshift3/ose-
docker-registry:latest*"}]}}}}'
```



注記

imagePolicyConfig 設定をインストールに追加するには、**openshift_master_image_policy_config** 変数を、以下のようにすべての **imagePolicyConfig** 設定を含む **json** でフォーマットされた文字列で使用できます。

```
openshift_master_image_policy_config={"imagePolicyConfig":
{"allowedRegistriesForImport":[{"domainName":"docker.io"},
{"domainName":"\*.docker.io"},{"domainName":"*.redhat.com"},
{"domainName":"*my.registry.example.com*"}]}}
```

ImagePolicy についての詳細は、「**ImagePolicy** 受付プラグイン」のセクションを参照してください。

7.3.8. OpenShift Container Platform レジストリーの統合

OpenShift Container Platform をスタンドアロンのコンテナーレジストリーとしてインストールし、レジストリー機能のみを提供できるようにすることができます。これには、OpenShift Container Platform プラットフォームを実行するようにこのレジストリーを使用できる利点があります。

OpenShift Container Platform レジストリーについての詳細は、「OpenShift Container レジストリーのスタンドアロンデプロイメントのインストール」を参照してください。

OpenShift Container Platform レジストリーを有効にするには、直前のすべてのセクションが適用されます。OpenShift Container Platform の観点では、このスタンドアロンの OpenShift Container レジストリーは外部レジストリーとして処理されますが、これはマルチテナントレジストリーであり、OpenShift Container Platform の承認モデルが使用されるため、いくつかの追加タスクが必要になります。このレジストリーは新規プロジェクトを独自の環境に作成するのではなく、これが通信するよう設定された OpenShift Container Platform 内に作成するため、作成されるすべてのプロジェクトに影響を与えます。

7.3.8.1. レジストリープロジェクトのクラスターへの接続

レジストリーはレジストリー Pod と Web インターフェースを含む完全な OpenShift Container Platform 環境であるため、レジストリーに新規プロジェクトを作成するプロセスは、**`oc new-project`** または **`oc create`** コマンドラインを使用して実行されるか、または Web インターフェースを使って実行されます。

プロジェクトが作成されると、通常のサービスアカウント (**`builder`**、**`default`**、および **`deployer`**) が自動的に作成され、プロジェクト管理者ユーザーにはパーミッションが付与されます。「匿名」ユーザーを含め、異なるユーザーにイメージをプッシュ/プルする権限を付与できます。

すべてのユーザーがレジストリー内の新規プロジェクトからイメージをプルできるようにするなどのいくつかのユースケースがありますが、OpenShift Container Platform とレジストリー間に 1:1 の関係を持たせることを希望している場合で、ユーザーが特定のプロジェクトからイメージのプッシュおよびプルを実行できるようにする場合には、いくつかの手順を実行する必要があります。

警告

レジストリー Web コンソールはプル/プッシュ操作に使用されるトークンを表示しますが、ここに表示されるトークンはセッショントークンであるため、期限切れになります。特定のパーミッションを持つサービスアカウントを作成することにより、管理者はサービスアカウントのパーミッションを制限することができ、たとえばイメージのプッシュ/プルに異なるサービスアカウントを使用できるようにできます。その後はサービスアカウントトークンの期限切れが生じなくなるため、トークンの期限切れ、シークレットの再作成その他のタスクについて設定する必要がなくなります。

手順

1. 新規プロジェクトを作成します。

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

2. レジストリープロジェクトを作成します。

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

3. サービスアカウントをレジストリープロジェクトに作成します。

```
$ oc create serviceaccount <my_serviceaccount> -n <registry_project>
```

4. **registry-editor** ロールを使用してイメージのプッシュおよびプルのパーミッションを付与します。

```
$ oc adm policy add-role-to-user registry-editor -z
<my_serviceaccount> -n <registry_project>
```

プルパーミッションのみが必要な場合、**registry-viewer** ロールを使用できます。

5. サービスアカウントトークンを取得します。

```
$ TOKEN=$(oc sa get-token <my_serviceaccount> -n <registry_project>)
```

6. トークンをパスワードとして使用し、**dockercfg** シークレットを作成します。

```
$ oc create secret docker-registry <my_registry> \
  --docker-server=<myregistry.example.com> --docker-username=
<notused> --docker-password=${TOKEN} --docker-email=<me@example.com>
```

7. **dockercfg** シークレットを使用し、シークレットをプル操作を実行するサービスアカウントにリンクして、イメージをレジストリーからプルします。イメージをプルするためのデフォルトのサービスアカウント名は **default** です。

```
$ oc secrets link default <my_registry> --for=pull
```

8. S2I 機能を使用してイメージをプッシュする場合、**dockercfg** シークレットは S2I Pod にマウントされるため、これをビルドを実行する適切なサービスアカウントにリンクする必要があります。イメージをビルドするために使用されるデフォルトのサービスアカウントの名前は **builder** です。

```
$ oc secrets link builder <my_registry>
```

9. **buildconfig** では、シークレットをプッシュまたはプル操作用に指定する必要があります。

```
"type": "Source",
"sourceStrategy": {
    "from": {
        "kind": "DockerImage",
        "name": "
<myregistry.example.com/registry_project/my_image:stable>"
    },
    "pullSecret": {
        "name": "<my_registry>"
    },
...[OUTPUT ABBREVIATED]...
"output": {
    "to": {
        "kind": "DockerImage",
        "name": "
<myregistry.example.com/registry_project/my_image:latest>"
    },
    "pushSecret": {
```

```
        "name": "<my_registry>"
    },
...[OUTPUT ABBREVIATED]...
```

第8章 証明書の管理

証明書は、OpenShift Container Platform クラスターの存続期間中に、証明書のライフサイクルにおいてさまざまなフェーズをたどります。以下の手順では、ライフサイクルの各フェーズを管理する方法について説明しています。

8.1. アプリケーションの自己署名型証明書の CA で署名される証明書への切り替え

一部のアプリケーションテンプレートはアプリケーションからクライアントに直接提示される自己署名型の証明書を作成します。たとえば、デフォルトでは、OpenShift Container Platform Ansible インストーラーのデプロイメントプロセスの一環として、メトリクスデプロイヤーが自己署名型の証明書を作成します。

これらの自己署名型の証明書はブラウザーで認識されません。この問題を緩和するには、公的に署名された証明書を使用し、これを自己署名型の証明書でトラフィックを再暗号化できるように設定します。

1. 既存ルートを削除します。

```
$ oc delete route hawkular-metrics -n openshift-infra
```

ルートが削除された状態で、新規ルートで re-encrypt ストラテジーと共に使用される証明書は、既存のワイルドカードおよびメトリクスデプロイヤーで作成される自己署名型の証明書でアセンブルされる必要があります。以下の証明書が利用可能でなければなりません。

- ワイルドカード CA 証明書
- ワイルドカードプライベートキー
- ワイルドカード証明書
- Hawkular CA 証明書
  各証明書は、新規ルート用にファイルシステムのファイルとして利用可能である必要があります。

  以下のコマンドを実行して Hawkular CA を取得し、この CA をファイルに保存してください。

```
$ oc get secrets hawkular-metrics-certificate -n openshift-infra
\
  -o jsonpath='{.data.hawkular-metrics-ca\.certificate}' | base64
-d > hawkular-internal-ca.crt
```

2. ワイルドカードのプライベートキー、証明書、および CA 証明書を見つけます。それぞれを **wildcard.key**、**wildcard.crt**、および **wildcard.ca** などの別々のファイルに配置します。

3. 新規 re-encrypt ルートを作成します。

```
$ oc create route reencrypt hawkular-metrics-reencrypt \
          -n openshift-infra \
          --hostname hawkular-metrics.ocp.example.com \
          --key wildcard.key \
          --cert wildcard.crt \
          --ca-cert wildcard.ca \
          --service hawkular-metrics \
```

```
            --dest-ca-cert hawkular-internal-ca.crt
```