OpenShift Container Platform 3.10

クラスターのアップグレード

OpenShift Container Platform 3.10 クラスターのアップグレード

Last Updated: 2018-11-14

OpenShift Container Platform 3.10 クラスターのアップグレード

OpenShift Container Platform 3.10 クラスターのアップグレード

法律上の通知

概要

本書を活用した OpenShift Container Platform 3.10 クラスターへのアップグレード

目次

第1章 アップグレード方式およびストラテジー

1.1. クラスターのアップグレードについて

OpenShift Container Platform の新規バージョンがリリースされると、既存のクラスターをアップグレードして最新の拡張機能およびバグ修正を適用できます。これには、リリース 3.7 から 3.9 などの以前のマイナーバージョンからのアップグレードや、マイナーバージョン (3.9.z リリース) 内での非同期エラータ更新の適用が含まれます。最新の変更点を確認するには、『OpenShift Container Platform 3.10 リリースノート』を参照してください。

注記

メジャーバージョン間の「コアとなるアーキテクチャーの変更」により、OpenShift Enterprise 2 環境は OpenShift Container Platform 3 にアップグレードできず、新規インストールが必要になります。

OpenShift Container Platform のアップグレードプロセスでは、Ansible Playbook を使用して、クラスターのアップグレードに必要なタスクを自動化します。初期インストール時や前回の正常なアップグレードで使用した「インベントリーファイル」を使用して、アップグレード Playbook を実行する必要があります。この方法を使用する場合には、アップグレードストラテジー (インプレースアップグレード) か、Blue-Green デプロイメントのいずれかを選択できます。

特に記述がない限り、特定のメジャーバージョンに含まれるノードおよびマスターには、前方と後方の「マイナーバージョン 1 つずつ」と互換性があるので、クラスターのアップグレードはスムーズに実行できます。ただし、クラスター全体をアップグレードするのに、一致しないバージョンを必要以上の時間をかけて実行することはできません。

アップグレードの前に、OpenShift Container Platform サービスがすべて実行されていることを確認します。コントロールプレーンのアップグレードが失敗した場合は、マスターのバージョンをチェックし、すべてのバージョンが同じであることを確認します。マスターがすべて同じバージョンである場合、アップグレードを再実行します。バージョンが異なる場合は、マスターをバージョン付けされた低いバージョンのマスターに一致するようにダウングレードしてからアップグレードを再実行します。

1.2. アップグレードストラテジー

OpenShift Container Platform クラスターのアップグレードでは、インプレースアップグレードまたは Blue-Green デプロイメントの 2 つのストラテジーを使用できます。

1.2.1. インプレースアップグレード

インプレースアップグレードでは、クラスターのアップグレードは実行中の単一クラスターのすべてのホストで実行されます。これはまずマスターで実行され、次にノードで実行されます。Pod はノードのアップグレードの開始前にそのノードから退避し、他の実行中のノードで再作成されます。これは、ユーザーアプリケーションのダウンタイムを短縮するのに役立ちます。

1.2.2. Blue-Green デプロイメント

「Blue-Green デプロイメント」アップグレード方式はインプレース方式と同様のフローに従います。まずマスターおよび etcd サーバーがアップグレードされます。ただしこの方式では、新規ノードをインプレースでアップグレードするのではなく、新規ノード用の並列環境が作成されます。

この方式では、新規デプロイメントの検証後、管理者は古いノードセット (例:「Blue」デプロイメント) から新規セット (例:「Green」デプロイメント) にトラフィックを切り換えることができます。問題が検出される場合も、古いデプロイメントへのロールバックを簡単かつすぐに実行できます。

第2章 クラスターの自動インプレースアップグレードの実行

2.1. インプレースアップグレードの概要

標準の「クラスターインストール」を使用してインストールしており、使用したインベントリーファイルが利用できる場合には、アップグレード Playbook を使用して OpenShift クラスターのアップグレードプロセスを自動化できます。

OpenShift Container Platform 3.10 では、クラスターのアーキテクチャーに大きな変更がありました。この点については、『OpenShift Container Platform 3.10 リリースノート』で詳しく説明しています。以下のセクションでは、これらの特筆すべき技術的な変更点が、OpenShift Container Platform 3.9 からのアップグレードプロセスにどのような影響を与えるかを説明します。

2.1.1. ノードグループおよびホストマッピングの定義

OpenShift Container Platform 3.10 以降では、ノードの設定はマスターから「ブートストラップ」されるようになりました。ノードサービスが起動されると、ノードは、**kubeconfig** および他のノード設定ファイルが存在するかどうかをクラスターに参加する前に確認します。存在しない場合には、ノードはマスターから設定をプルしてから、クラスターに参加します。

このプロセスが導入され、管理者はノードホストごとに一意にノード設定を手動でメンテナンスする必要がなくなり、ノードホストの **/etc/origin/node/node-config.yaml** ファイルの内容がマスターから取得する ConfigMaps により提供されるようになりました。

どのノードホストにどの ConfigMap を使用するかをマッピングするには、インベントリーファイルにホストごとに **ノードグループ** を定義してから設定する必要があります。これらの手順は、「自動化アップグレードの準備」のセクションで取り扱います。

ノードの ConfigMaps
ノード設定の定義用の ConfigMaps は **openshift-node** プロジェクトで利用できる必要があります。ConfigMaps はノードラベルの信頼できる定義となり、以前の **`openshift_node_labels`** の値は事実上、無視されます。

インストーラーには Playbook が含まれており、以下のデフォルトの ConfigMaps を作成します。

- **`node-config-master`**
- **`node-config-infra`**
- **`node-config-compute`**

以下の ConfigMaps も作成され、複数のロールにノードをラベル付けします。

- **`node-config-all-in-one`**
- **`node-config-master-infra`**

重要

ノードホストの **/etc/origin/node/node-config.yaml** ファイルには変更を加えないようにしてください。このファイルは、ノードが使用する ConfigMap に定義されている設定により上書きされます。

ノードグループの定義

openshift-ansible パッケージのアップグレード後に、ノードグループ定義のデフォルトセットが **/usr/share/ansible/openshift-ansible/roles/openshift_facts/defaults/main.yml** ファイルの YAML 形式ではどのようになっているかを確認できます。

```
openshift_node_groups:
  - name: node-config-master (1)
    labels:
      - 'node-role.kubernetes.io/master=true' (2)
    edits: [] (3)
  - name: node-config-infra
    labels:
      - 'node-role.kubernetes.io/infra=true'
    edits: []
  - name: node-config-compute
    labels:
      - 'node-role.kubernetes.io/compute=true'
    edits: []
  - name: node-config-master-infra
    labels:
      - 'node-role.kubernetes.io/infra=true,node-
role.kubernetes.io/master=true'
    edits: []
  - name: node-config-all-in-one
    labels:
      - 'node-role.kubernetes.io/infra=true,node-
role.kubernetes.io/master=true,node-role.kubernetes.io/compute=true'
    edits: []
```

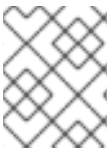
(1) ノードのグループ名

(2) ノードのグループに関連付けられるノードラベルの一覧

(3) ノードグループの設定に対する編集

インベントリーファイルの **[OSEv3:vars]** グループに **openshift_node_groups** 変数が設定されていない場合には、上記で定義したデフォルト値が使用されます。ただし、これらのデフォルト値とは違う値を使用する場合には、インベントリーファイルに (任意の全ノードグループなど) **openshift_node_groups** の構造全体を定義する必要があります。

openshift_node_groups の値は、デフォルト値とマージされないので、YAML 定義を先に Python ディクショナリーに変換する必要があります。その後に、**edits** フィールドを使用して、鍵と値のペアを指定して任意のノード設定変数に変更できます。



注記

設定可能なノード変数に関する情報は、「マスターとノード設定ファイル」を参照してください。

たとえば、インベントリーファイルの以下のエントリーは、**node-config-master**、**node-config-infra** および **node-config-compute** という名前のグループを定義し、**node-config-compute** の ConfigMap を編集して **kubeletArguments.pods-per-core** を **20** に設定します。

```
openshift_node_groups=[{'name': 'node-config-master', 'labels': ['node-
```

```
role.kubernetes.io/master=true']}, {'name': 'node-config-infra', 'labels':
['node-role.kubernetes.io/infra=true',]}, {'name': 'node-config-compute',
'labels': ['node-role.kubernetes.io/compute=true'], 'edits': [{ 'key':
'kubeletArguments.pods-per-core','value': ['20']}]}]
```

openshift_node_group.yml Playbook が実行されるたびに、**edits** フォールドで定義した変更により、関連の ConfigMap (この例では **node-config-compute**) が更新され、最終的にホスト上のノードの設定ファイルに影響を与えます。

2.1.2. アップグレードのワークフロー

3.9 から 3.10 へのコントロールプレーンの自動アップグレードでは、以下の手順が実行されます。

- 復元の目的ですべての etcd データのバックアップを作成する。
- API およびコントローラーを 3.9 から 3.10 に更新する。
- 内部データ構造を 3.10 に更新する。
- デフォルトルーター (ある場合) を 3.9 から 3.10 に更新する。
- デフォルトレジストリー (ある場合) を 3.9 から 3.10 に更新する。
- デフォルトイメージストリームおよび InstantApp テンプレートを更新する。

3.9 から 3.10 へのノードの自動アップグレードではノードのローリングアップデートを実行し、以下が行われます。

- ノードのサブセットにスケジュール対象外 (unschedulable) のマークを付け、それらのノードを Pod からドレイン (解放) します。
- 3.9 から3.10 にノードコンポーネントを更新します。
- ローカルの設定から、ブートストラップされた TLS および Config に変換します。
- **systemd** サービスから DaemonSets に SDN コンポーネントを変換します。
- それらのノードをサービスに返します。

重要

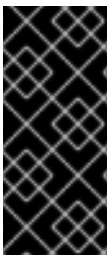
- アップグレードに進む前に、すべての「前提条件」を満たしていることを確認します。前提条件を満たしていないと、アップグレードが失敗する可能性があります。
- GlusterFS を使用している場合は、「コンテナー化された GlusterFS を使用する場合の特別な考慮事項」を参照してからアップグレードを行ってください。
- GCE 永続ディスク (gcePD) を使用している場合は、「gcePD を使用する場合の特別な考慮事項」を参照してからアップグレードを行ってください。
- アップグレードの前日に、OpenShift Container Platform ストレージの移行を検証して、サービスを停止する前に考えられる問題を解決しておくようにしてください。

```
$ oc adm migrate storage --include=* --loglevel=2 --
confirm --config /etc/origin/master/admin.kubeconfig
```

自動化されたアップグレード Playbook は、インベントリーファイルと共に **ansible-playbook** コマンドを使用して Ansible で直接実行されます。これは通常インストール (advanced installation) 方式と同様です。以下のいずれのシナリオでも、同じ **v3_10** アップグレード Playbook を使用することができます。

- 既存の OpenShift Container Platform 3.9 クラスターの 3.10 へのアップグレード
- 既存の OpenShift Container Platform 3.9 クラスターの最新の「非同期エラータ更新」へのアップグレード

2.2. 自動アップグレードの準備



重要

OpenShift Container Platform 3.10 にクラスターをアップグレードする前に、クラスターを「バージョン 3.9 の最新の非同期リリース」にアップグレードしておく必要があります。クラスターが 3.9 以前のバージョンの場合には、先に 1 バージョンごとにアップグレードしてください。たとえば、3.6 から 3.7 にアップグレードし、3.7 から 3.9 (バージョン 3.8 は「省略されました」) にアップグレードしてください。

注記

アップグレードを試行する前に、「環境ヘルスチェック」のガイダンスに従い、クラスターの健全性を確認します。これにより、ノードが **Ready** 状態で、必要とされる開始バージョンを実行していることが分かり、診断エラーまたは警告がないことを確認できます。

自動アップグレードを準備するには、以下を実行します。

1. RHSM から最新のサブスクリプションデータをプルします。

```
# subscription-manager refresh
```

2. OpenShift Container Platform 3.9 から 3.10 にアップグレードする場合には、以下を行います。

a. 後で OpenShift Container Platform 3.9 にダウングレードする場合に必要となる以下のファイルをバックアップします。

   i. マスターホストで、以下のファイルをバックアップします。

```
/usr/lib/systemd/system/atomic-openshift-master-api.service
/usr/lib/systemd/system/atomic-openshift-master-
controllers.service
/etc/sysconfig/atomic-openshift-master-api
/etc/sysconfig/atomic-openshift-master-controllers
/etc/origin/master/master-config.yaml
/etc/origin/master/scheduler.json
```

   コントロールプレーンの静的 Pod のアーキテクチャーに変更が加えられるので、**systemd** ファイルはアップグレード時に削除されます。

   ii. ノードのホストで (ノードのコンポーネントが含まれるマスターなど)、以下のファイルをバックアップします。

```
/usr/lib/systemd/system/atomic-openshift-*.service
/etc/origin/node/node-config.yaml
```

   iii. etcd ホストで (etcd が共存するマスターなど)、以下のファイルをバックアップします。

```
/etc/etcd/etcd.conf
```

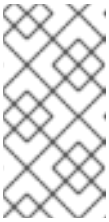
b. アップグレードプロセスでは、復元の目的で全 etcd データのバックアップを作成しますが、**/backup/etcd-xxxxxx/backup.db** に最新の etcd のバックアップがあることを確認してから続行してください。etcd の手動でのバックアップ手順は、『Day 2 操作ガイド』に記載されています。

c. 手作業で、各マスターおよびノードホストの 3.9 リポジトリーを無効にして、3.10 レポジトリーを有効にします。**rhel-7-server-ansible-2.4-rpms** リポジトリーをまだ有効にしていない場合には、有効にしてください。

```
# subscription-manager repos --disable="rhel-7-server-ose-3.9-
rpms" \
    --enable="rhel-7-server-ose-3.10-rpms" \
    --enable="rhel-7-server-rpms" \
    --enable="rhel-7-server-extras-rpms" \
    --enable="rhel-7-server-ansible-2.4-rpms" \
    --enable="rhel-7-fast-datapath-rpms"
# yum clean all
```

d. アップグレード Playbook を実行するホストで、最新版の **openshift-ansible** パッケージが設定されているようにしてください。

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



注記

以前の OpenShift Container Platform リリースでは、この手順で **atomic-openshift-utils** パッケージがインストールされていましたが、OpenShift Container Platform 3.10 以降ではこのパッケージが削除され、**openshift-ansible** パッケージが必要な項目をすべて提供します。

e. インベントリーファイルの **[OSEv3:vars]** グループに **openshift_node_groups** 変数が設定されていない場合には、**openshift_node_group.yml** playbook の実行時に、ノードグループと ConfigMaps のデフォルトセットが作成されます。デフォルト値と違う値を使用する場合には、「ノードグループとホストマッピングの定義」に記載されているように、Python ディクショナリー形式を使用して全ノードグループセットを定義し、以下のように **name**、**labels** および **edits** を指定して ConfigMaps を変更します。

```
[OSEv3:vars]

openshift_node_groups=[{'name': 'node-config-master', 'labels':
['node-role.kubernetes.io/master=true']}, {'name': 'node-config-
infra', 'labels': ['node-role.kubernetes.io/infra=true',]},
{'name': 'node-config-compute', 'labels': ['node-
role.kubernetes.io/compute=true'], 'edits': [{ 'key':
'kubeletArguments.pods-per-core','value': ['20']}]}]
```

f. OpenShift Container Platform 3.9 クラスターが新規ノードグループの定義とマッピングを使用するように変換するには、**[nodes]** インベントリーグループで以前に定義した全ホストに **openshift_node_group_name** を割り当てる必要があります。この値を使用して、各ノードを設定する ConfigMap を選択します。
例:

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

さらに、**openshift_node_labels** 設定がある場合には、既存のホストエントリーの **[nodes]** グループから削除します。ノードラベルは、ホストの **openshift_node_group_name** と関連付けられている ConfigMap で定義されるはずです。

g. アップグレードプロセスは、**openshift-node** プロジェクトのノードをブートストラップするのに必要な新しい ConfigMaps を配置するまでブロックされます。以下の Playbook を実行して作成したデフォルト値を使用します (または、以前の手順で **openshift_node_groups** の構造を定義した場合には、カスタムのセットが作成されます)。

```
# ansible-playbook -i </path/to/inventory/file> \
    /usr/share/ansible/openshift-ansible/playbooks/openshift-
master/openshift_node_group.yml
```

h. 各 ConfigMap が想定どおりに作成、定義されていることを慎重に検証します。**openshift-node** プロジェクトの全 ConfigMaps 一覧を取得します。

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

次に **oc describe** を使用して個別に検証します。

```
$ oc describe configmaps -n openshift-node <configmap_name>
```

3. (初期インストールまたは最近のクラスターのアップグレードなどで) Ansible Playbook を最後に実行した後にマスターまたはノード設定ファイルに設定変更を手動で加えた場合は、「Ansible インベントリーファイルの設定」を参照してください。手動で加えた変更に関連する変数については、アップグレードの実行前に同等の変更をインベントリーファイルに適用してください。これを実行しないと、手動による変更がアップグレード時にデフォルト値で上書きされ、Pod が適切に実行されなくなるか、または他のクラスターの安定性に問題が生じる可能性があります。
とくに、マスター設定ファイルの **admissionConfig** 設定に変更を加えた場合は、「Ansible インベントリーファイルの設定」で **openshift_master_admission_plugin_config** 変数を確認してください。これを確認しない場合、**ClusterResourceOverride** 設定を事前に手動で設定している場合には (「マスターでのオーバーコミットの設定」など)、Pod が **Pending** 状態のままになる可能性があります。

上記を確認した後に、以下のセクションでアップグレードプロセスのしくみをさらに確認し、カスタマイズする場合はアップグレードの追加のカスタマイズオプションについて決定します。アップグレードを実行する準備が整ったら、「最新の OpenShift Container Platform 3.10 リリースへのアップグレード」に進んでください。

2.2.1. コントロールプレーンとノードの別フェーズでのアップグレード

OpenShift Container Platform クラスターは 1 つまたは複数のフェーズでアップグレードできます。単一の Ansible Playbook を実行してすべてのホストを 1 つのフェーズでアップグレードすることも、別々の Playbook を使用して **コントロールプレーン** (マスターコンポーネント) およびノードを複数のフェーズでアップグレードすることもできます。

注記

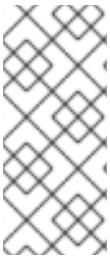
完全なアップグレードプロセスおよびこれらの Playbook を呼び出すタイミングについては、「最新の OpenShift Container Platform 3.10 リリースへのアップグレード」に説明されています。

複数のフェーズでアップグレードする場合、コントロールプレーンのフェーズには、以下のアップグレードが含まれます。

- マスターコンポーネント
- マスターで実行されるノードサービス
- マスターで実行される Docker または CRI-O
- スタンドアロン etcd ホストで実行される Docker または CRI-O

ノードのみをアップグレードする場合、コントロールプレーンはすでにアップグレードされている必要があります。ノードのフェーズには、以下のアップグレードが含まれます。

- スタンドアロンノードで実行されるノードサービス
- スタンドアロンノードで実行される Docker または CRI-O

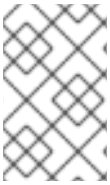


注記

マスターコンポーネントを実行するノードは、ノードサービスや Docker が実行されている場合でもノードのアップグレードフェーズには含まれず、それらはコントロールプレーンのアップグレードフェーズの一環としてアップグレードされます。これは、マスターのノードサービスおよび Docker のアップグレードが 2 回 (1 回目はコントロールプレーンのフェーズ、2 回目はノードのフェーズ) 実行されないようにします。

2.2.2. ノードのアップグレードのカスタマイズ

アップグレードを単一または複数フェーズのいずれで実行する場合でも、**-e** オプションを使って特定の Ansible 変数をアップグレード Playbook に渡すことで、アップグレードのノード部分の処理方法をカスタマイズできます。



注記

完全なアップグレードプロセスおよびこれらの Playbook を呼び出すタイミングについては、「最新の OpenShift Container Platform 3.10 リリースへのアップグレード」に説明されています。

openshift_upgrade_nodes_serial 変数は整数またはパーセンテージに設定して同時にアップグレードできるノードホストの数を制御できます。デフォルトは **1** であり、この場合ノードは一度に 1 つずつアップグレードされます。

たとえば、一度に検出されるノードの全体数の 20 パーセントをアップグレードするには、以下を実行します。

```
$ ansible-playbook -i <path/to/inventory/file> \
    </path/to/upgrade/playbook> \
    -e openshift_upgrade_nodes_serial="20%"
```

openshift_upgrade_nodes_label 変数を指定すると、特定のラベルを持つノードのみをアップグレードするように指定できます。これは **openshift_upgrade_nodes_serial** 変数と組み合わせて使用することもできます。

たとえば、**group1** リージョンのノードのみを一度に 2 つアップグレードするには、以下を実行します。

```
$ ansible-playbook -i <path/to/inventory/file> \
    </path/to/upgrade/playbook> \
    -e openshift_upgrade_nodes_serial="2" \
    -e openshift_upgrade_nodes_label="region=group1"
```

注記

ノードラベルに関する詳細は、「ノードの管理」を参照してください。

openshift_upgrade_nodes_max_fail_percentage 変数を使用すると、各バッチで失敗するノードの数を指定できます。失敗のパーセンテージは Playbook がアップグレードを中止する前に指定した値を上回っている必要があります。

openshift_upgrade_nodes_drain_timeout 変数を使用すると、終了前の待機時間の長さを指定できます。

以下の例では、一度に 10 ノードをアップグレードして、20 パーセントを超えるノードが失敗する場合にはアップグレードが中断されます。ノードをドレイン (解放) するまでに 600 秒の待機時間があります。

```
$ ansible-playbook -i <path/to/inventory/file> \
    </path/to/upgrade/playbook> \
    -e openshift_upgrade_nodes_serial=10 \
    -e openshift_upgrade_nodes_max_fail_percentage=20 \
    -e openshift_upgrade_nodes_drain_timeout=600
```

2.2.3. Ansible Hook を使用したアップグレードのカスタマイズ

OpenShift Container Platform のアップグレード時の特定の操作の実行中に、**Hook** というシステムでカスタムタスクを実行できます。Hook を使うと、管理者はアップグレード時の特定分野の前後に実行するタスクを定義するファイルを指定できます。これは、OpenShift Container Platform のアップグレード時にカスタムインフラストラクチャーを検証し、変更するのに非常に便利です。

Hook が失敗すると操作も失敗することに留意してください。つまり、Hook が適切であると複数回実行でき、同じ結果を出せます。適切な Hook にはべき等性があります。

2.2.3.1. 制限

- Hook には定義され、バージョン付けされたインターフェースがありません。Hook は内部の **openshift-ansible** 変数を使用できますが、それらの変数が今後のリリースでもそのまま残される保証はありません。また、Hook は今後バージョン付けされる可能性があります。これは Hook が最新の **openshift-ansible** と連動するように更新する必要があることを示す警告となります。

- Hook にはエラー処理機能がなく、Hook にエラーが生じると、アップグレードプロセスは中止します。その場合、問題に対応してからアップグレードを再実行する必要があります。

2.2.3.2. Hook の使用

Hook は **ホスト** インベントリーファイルの **OSEv3:vars** セクションに定義されます。

各 Hook は Ansible タスクを定義する YAML ファイルをポイントする必要があります。このファイルは **include** として使用されます。つまり、このファイルは Playbook にすることはできず、タスクのセットである必要があります。あいまいさを避けるために Hook ファイルへの絶対パスを使用することがベストプラクティスとして推奨されます。

インベントリーファイルの Hook 定義の例

```
[OSEv3:vars]
openshift_master_upgrade_pre_hook=/usr/share/custom/pre_master.yml
openshift_master_upgrade_hook=/usr/share/custom/master.yml
openshift_master_upgrade_post_hook=/usr/share/custom/post_master.yml

openshift_node_upgrade_pre_hook=/usr/share/custom/pre_node.yml
```

```
openshift_node_upgrade_hook=/usr/share/custom/node.yml
openshift_node_upgrade_post_hook=/usr/share/custom/post_node.yml
```

pre_master.yml タスクの例

```
---
# Trivial example forcing an operator to ack the start of an upgrade
# file=/usr/share/custom/pre_master.yml

- name: note the start of a master upgrade
  debug:
      msg: "Master upgrade of {{ inventory_hostname }} is about to start"

- name: require an operator agree to start an upgrade
  pause:
      prompt: "Hit enter to start the master upgrade"
```

2.2.3.3. 利用可能なアップグレードフック

表2.1 マスターアップグレードのフック

フック名	説明
`openshift_master_upgrade_pre_hook`	• 各マスターのアップグレード **前** に実行します。 • このフックは **各マスター** に対して連続して実行されます。 • タスクが異なるホストに対して実行される必要がある場合、該当するタスクは `delegate_to` または `local_action` を使用する必要があります。
`openshift_master_upgrade_hook`	• 各マスターのアップグレード **後** に実行しますが、そのサービスまたはシステムの再起動 **前** に実行します。 • このフックは **各マスター** に対して連続して実行されます。 • タスクが異なるホストに対して実行される必要がある場合、該当するタスクは `delegate_to` または `local_action` を使用する必要があります。

フック名	説明
`openshift_master_upgrade_post_hook`	• 各マスターをアップグレードし、そのサービスまたはシステムが再起動した **後** に実行します。 • このフックは **各マスター** に対して連続して実行されます。 • タスクが異なるホストに対して実行される必要がある場合、該当するタスクは `delegate_to` または `local_action` を使用する必要があります。

表2.2 ノードアップグレードのフック

フック名	説明
`openshift_node_upgrade_pre_hook`	• 各ノードのアップグレード **前** に実行します。 • このフックは **ノードごと** に順に実行されます。 • タスクが異なるホストに対して実行される必要がある場合、該当するタスクは `delegate_to` または `local_action` を使用する必要があります。
`openshift_node_upgrade_hook`	• 各ノードのアップグレード **後** で、かつ、もう一度スケジューリング可能とマークされる **前** に実行します。 • このフックは **ノードごと** に順に実行されます。 • タスクが異なるホストに対して実行される必要がある場合、該当するタスクは `delegate_to` または `local_action` を使用する必要があります。

フック名	説明
`openshift_node_upgrade_post_hook`	• 各ノードのアップグレード **後** に実行します。これは、ノードのアップグレードアクションの中で **最後** となります。 • このフックは **ノードごと** に順に実行されます。 • タスクが異なるホストに対して実行される必要がある場合、該当するタスクは `delegate_to` または `local_action` を使用する必要があります。

2.3. 最新の OPENSHIFT CONTAINER PLATFORM 3.10 リリースへのアップグレード

既存の OpenShift Container Platform 3.9 または 3.10 クラスターを最新の 3.10 リリースにアップグレードするには、以下を実行します。

1. 「自動化アップグレードの準備」のステップを満たしていることを確認した上で、最新のアップグレード Playbook を使用していることを確認します。

2. インベントリーファイルの **openshift_deployment_type** パラメーターが **openshift-enterprise** に設定されていることを確認します。

3. ホストのローリングおよび完全なシステムの再起動を実行する必要がある場合は、インベントリーファイルの **openshift_rolling_restart_mode** パラメーターを **system** に設定できます。これを設定しないと、デフォルト値の **services** が HA マスターのローリングサービスの再起動を実行しますが、システムの再起動は実行しません。詳細については、「クラスター変数の設定」を参照してください。

4. この時点で、アップグレードを単一フェーズで実行するか、または複数フェーズで実行するかを選択できます。各フェーズでアップグレードされるコンポーネントについての詳細は、「コントロールプレーンとノードの別フェーズでのアップグレード」を参照してください。
インベントリーファイルがデフォルトの **/etc/ansible/hosts** 以外の場所に置かれている場合、**-i** フラグを追加してその場所を指定します。以前に **atomic-openshift-installer** コマンドを使用してインストールを実行したことがある場合は、必要に応じて **~/.config/openshift/hosts** で最後に使用されたインベントリーファイルを確認できます。

   - **オプション A**: 単一フェーズでコントロールプレーンおよびノードをアップグレードします。
   upgrade.yml Playbook を実行し、1 つの Playbook を使用して単一フェーズでクラスターのアップグレードを実行します。ここでも、まずコントロールプレーンをアップグレードしてからノードのインプレースアップグレードが実行されます。

     ```
     # ansible-playbook -i </path/to/inventory/file> \
         /usr/share/ansible/openshift-ansible/playbooks/byo/openshift-
     cluster/upgrades/v3_10/upgrade.yml
     ```

   - **オプション B**: 別々のフェーズでコントロールプレーンおよびノードをアップグレードします。

a. コントロールプレーンのみを実行するには、**upgrade_control_plane.yaml** Playbook を実行します。

```
# ansible-playbook -i </path/to/inventory/file> \
    /usr/share/ansible/openshift-
ansible/playbooks/byo/openshift-
cluster/upgrades/v3_10/upgrade_control_plane.yml
```

b. ノードのみを実行するには、**upgrade_nodes.yaml** Playbook を実行します。

```
# ansible-playbook -i </path/to/inventory/file> \
    /usr/share/ansible/openshift-
ansible/playbooks/byo/openshift-
cluster/upgrades/v3_10/upgrade_nodes.yml \
    [-e <customized_node_upgrade_variables>] 1
```

1 必要な **<customized_node_upgrade_variables>** については、「ノードのアップグレードのカスタマイズ」を参照してください。

「ノードのアップグレードのカスタマイズ」で説明されているようにノードをグループでアップグレードしている場合、すべてのノードが正常にアップグレードされるまで **upgrade_nodes.yml** Playbook の起動を継続します。

5. すべてのマスターおよびノードのアップグレードの完了後に、すべてのホストを再起動します。再起動後に追加の機能が有効化されていない場合は、「アップグレードの検証」を実行できます。追加機能が有効化されている場合には、次の手順は以前に有効にした追加機能の内容によって変わります。

機能	次の手順
集計されたロギング	EFK ロギングスタックのアップグレード
クラスターメトリクス	クラスターメトリクスのアップグレード

2.4. EFK ロギングスタックのアップグレード

既存の EFK ロギングスタックデプロイメントをアップグレードするには、提供されている **/usr/share/ansible/openshift-ansible/playbooks/openshift-logging/config.yml** Ansible Playbook を使用する必要があります。これは既存のクラスターでロギングを最初にデプロイする際に使用する Playbook ですが、既存のロギングデプロイメントをアップグレードする場合にも使用されます。

1. これをまだ実行していない場合は、「コンテナーログの集計」のトピックの「ロギング Ansible 変数の指定」を参照し、Ansible インベントリーファイルを更新して少なくとも **[OSEv3:vars]** セクション内に以下の必要な変数を設定します。

```
[OSEv3:vars]

openshift_logging_install_logging=true 1
```

1 ロギングスタックをアップグレードする機能を有効にします。

2. 「ロギング Ansible 変数の指定」で説明されているように、デフォルトを上書きするために指定する必要のあるその他の **openshift_logging_*** 変数を追加します。

3. インベントリーファイルの更新が終了したら、「EFK スタックのデプロイ」の説明に従って **openshift-logging/config.yml** Playbook を実行し、ロギングデプロイメントのアップグレードを完了します。

注記

EFK コンポーネントの Fluentd DeploymentConfig および DaemonSet が次のように設定されている場合、以下に注意してください。

```
image: <image_name>:<vX.Y>
imagePullPolicy: IfNotPresent
```

最新バージョンの **<image_name>** は、Pod が再デプロイされるノードに同じ **<image_name:vX.Y>** のイメージがローカルに保存されている場合にはプルされない可能性があります。その場合は、DeploymentConfig および DaemonSet を手動で **imagePullPolicy: Always** に設定し、再度プルされるようにします。

2.5. クラスターメトリクスのアップグレード

既存のクラスターメトリクスのデプロイメントをアップグレードするには、提供されている **/usr/share/ansible/openshift-ansible/playbooks/openshift-metrics/config.yml** Ansible Playbook を使用する必要があります。これは、既存クラスターでメトリクスを最初にデプロイしている場合に使用する Playbook ですが、既存のメトリクスデプロイメントをアップグレードする場合にも使用されます。

1. これをまだ実行していない場合は、「クラスターメトリクスの有効化」のトピックの「メトリクス Ansible 変数の指定」を参照し、Ansible インベントリーファイルを更新して少なくとも **[OSEv3:vars]** セクション内で以下の必要な変数を設定します。

```
[OSEv3:vars]

openshift_metrics_install_metrics=true (1)
openshift_metrics_hawkular_hostname=<fqdn> (2)
openshift_metrics_cassandra_storage_type=(emptydir|pv|dynamic) (3)
```

(1) メトリクスデプロイメントをアップグレードする機能を有効にします。

(2) Hawkular Metrics ルートに使用します。完全修飾ドメイン名を指定します。

(3) 以前のデプロイメントと同じタイプを選択します。

2. 「メトリクス Ansible 変数の指定」に説明されているように、デフォルトを上書きするために指定する必要のあるその他の **openshift_metrics_*** 変数を追加します。

3. インベントリーファイルの更新が終了したら、「メトリクスコンポーネントのデプロイ」の説明に従って **openshift-metrics/config.yml** Playbook を実行し、メトリクスデプロイメントのアップグレードを完了します。

2.6. 混在環境についての特別な考慮事項

混在環境のアップグレード (Red Hat Enterprise Linux および Red Hat Enterprise Linux Atomic Host を含む環境など) では、**openshift_pkg_version** および **openshift_image_tag** の両方を設定する必要があります。混在環境では、**openshift_pkg_version** のみを指定する場合、その番号が Red Hat Enterprise Linux および Red Hat Enterprise Linux Atomic Host のイメージに使用されます。

2.7. コンテナー化された GLUSTERFS を使用する場合の特別な考慮事項

OpenShift Container Platform のアップグレード時に、GlusterFS Pod が実行されているノードのセットをアップグレードする必要があります。

drain および **unschedule** は daemonset の一部として実行されているために GlusterFS Pod を停止せず、退避しません。そのため、これらのノードをアップグレードする際には特別な考慮が必要になります。

また、複数のノードでアップグレードを同時に実行される可能性もあり、これにより複数のノードが GlusterFS Pod をホストしている場合にデータ可用性の問題が生じる可能性があります。

シリアルアップグレードが実行されている場合でも、次のノードの GlusterFS が終了する前に GlusterFS が自動修復操作のすべてを完了するのに必要な時間が確保される保証はありません。そのため、クラスターの状態が正常でなくなるか、または不明な状態になる可能性があります。したがって、以下の手順を実行することをお勧めします。

1. コントロールプレーンをアップグレードします (マスターノードおよび etcd ノード)。
2. 標準 **infra** ノード (ルーター、レジストリー、ロギング、およびメトリクス) をアップグレードします。

注記

これらのグループのノードのいずれかが GlusterFS を実行している場合、この手順のステップ 4 を同時に実行します。GlusterFS ノードは (**app** や **infra** などの) クラス内の他のノードと共に 一度に 1 つずつアップグレードする必要があります。

3. アプリケーションコンテナーを実行する標準ノードをアップグレードします。

注記

これらのグループのノードのいずれかが GlusterFS を実行している場合、この手順のステップ 4 を同時に実行します。GlusterFS ノードは (**app** や **infra** などの) クラス内の他のノードと共に 一度に 1 つずつアップグレードする必要があります。

4. GlusterFS を実行する OpenShift Container Platform ノードを一度に 1 つずつアップグレードします。
   a. **oc get daemonset** を実行して **NODE-SELECTOR** にあるラベルを検証します。デフォルト値は **storagenode=glusterfs** です。
   b. daemonset ラベルをノードから削除します。

```
$ oc label node <node_name> <daemonset_label>-
```

   これにより、GlusterFS Pod がこのノード上で終了します。

c. 追加のラベル (**type=upgrade** など) をアップグレードする必要のあるノードに追加します。

d. アップグレード Playbook を GlusterFS を終了した単一ノードで実行するには、**-e openshift_upgrade_nodes_label="type=upgrade"** を使用します。

e. アップグレードの完了時に、deamonset セレクターでノードのラベルを変更します。

```
$ oc label node <node_name> <daemonset_label>
```

f. GlusterFS Pod が再生成され、表示されるまで待機します。

g. **oc rsh** を Pod に実行し、すべてのボリュームが自動修復されていることを確認します。

```
$ oc rsh <GlusterFS_pod_name>
$ for vol in `gluster volume list`; do gluster volume heal $vol
info; done
```

すべてのボリュームが自動修復され、未処理のタスクがないことを確認します。**heal info** コマンドは所定ボリュームの自動修復プロセスのすべての未処理エントリーを一覧表示します。ボリュームは、ボリュームの **Number of entries** が **0** の場合に自動修正済みとみなされます。

h. アップグレードラベル (**type=upgrade** など) を削除し、次の GlusterFS ノードに移動します。

2.8. GCEPD を使用する場合の特別な考慮事項

デフォルトの gcePD ストレージプロバイダーは RWO (Read-Write Only) アクセスモードを使用するため、レジストリーでローリングアップグレードを実行したり、レジストリーを複数 Pod に拡張したりすることはできません。そのため、OpenShift Container Platform のアップグレード時に以下の環境変数を Ansible インベントリーファイルに指定する必要があります。

```
[OSEv3:vars]

openshift_hosted_registry_storage_provider=gcs
openshift_hosted_registry_storage_gcs_bucket=bucket01
openshift_hosted_registry_storage_gcs_keyfile=test.key
openshift_hosted_registry_storage_gcs_rootdirectory=/registry
```

2.9. アップグレードの検証

以下を確認します。

- クラスターが正常である。
- マスター、ノード、etcd サービスまたは静的 Pod が正常に実行されている。
- OpenShift Container Platform、**docker-registry**、およびルーターバージョンが正しい。
- 元のアプリケーションが依存として利用可能な状態で、新規アプリケーションの作成が可能である。
- **oc adm diagnostics** を実行してもエラーが生じない。

アップグレードを検証するには、以下を確認します。

1. すべてのノードに **Ready** のマークが付けられていることを確認します。

```
# oc get nodes
NAME                      STATUS    ROLES        AGE       VERSION
master1.example.com       Ready     master       47d
v1.10.0+b81c8f8
master2.example.com       Ready     master       47d
v1.10.0+b81c8f8
master3.example.com       Ready     master       47d
v1.10.0+b81c8f8
infra-node1.example.com   Ready     infra        47d
v1.10.0+b81c8f8
infra-node2.example.com   Ready     infra        47d
v1.10.0+b81c8f8
node1.example.com         Ready     compute      47d
v1.10.0+b81c8f8
node2.example.com         Ready     compute      47d
v1.10.0+b81c8f8
```

2. コントロールプレーンの静的 Pod が実行されていることを確認します。

```
# oc get pods -n kube-system
NAME                                  READY     STATUS    RESTARTS
AGE
master-api-master1.example.com            1/1       Running   4
1h
master-controllers-master1.example.com    1/1       Running   3
1h
master-etcd-master1.example.com           1/1       Running   6
5d
[...]
```

3. 必要なバージョンの **docker-registry** および **router** イメージを実行していることを確認します (デプロイされている場合)。

```
# oc get -n default dc/docker-registry -o json | grep \"image\"
    "image": "openshift3/ose-docker-registry:v3.10",
# oc get -n default dc/router -o json | grep \"image\"
    "image": "openshift3/ose-haproxy-router:v3.10",
```

4. マスター上で診断ツールを使用し、一般的な問題を検索します。

```
# oc adm diagnostics
...
[Note] Summary of diagnostics execution:
[Note] Completed with no errors or warnings seen.
```

第3章 BLUE-GREEN デプロイメント

3.1. 概要

注記

このトピックでは、インプレースアップグレード方法に代わるノードホストのアップグレード方法について説明します。

Blue-Green デプロイメント のアップグレード方式はインプレース方式と同様のフローに基づいて実行されます。この場合もマスターおよび etcd サーバーが最初にアップグレードされます。ただし、インプレースアップグレードを実行する代わりに、新規ノードホストの並列環境が作成されます。

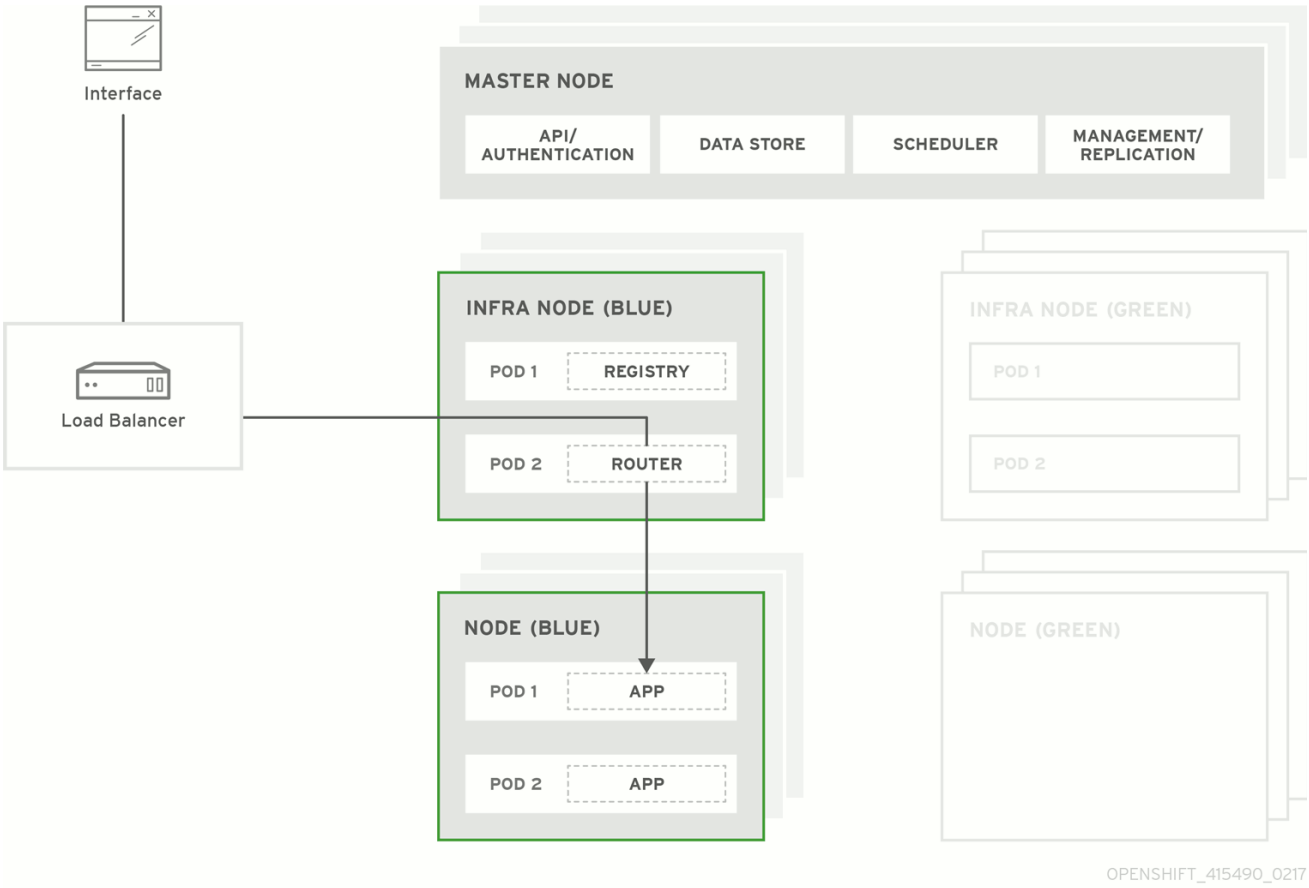
この方式では、管理者は新規デプロイメントの検証後、古いノードホストセット (例:「**Blue**」デプロイメント) から新規セット (例:「**Green** デプロイメント) にトラフィックを切り換えることができます。問題が検出される場合も、古いデプロイメントへのロールバックを簡単かつすぐに実行できます。

Blue-Green がソフトウェアについてのデプロイの証明済みの有効なストラテジーであるものの、トレードオフも常にあります。すべての環境が Blue-Green デプロイメントの適切な実行に必要な同じアップタイム要件を満たしている訳でもなく、これに必要なリソースがある訳でもありません。

OpenShift Container Platform 環境では、Blue-Green デプロイメントに最も適した候補はノードホストです。すべてのユーザープロセスはこれらのシステムで実行され、OpenShift Container Platform インフラストラクチャーの重要な部分もこれらのリソースで自己ホストされます。アップタイムはこれらのワークロードで最も重要であり、Blue-Green デプロイメントによって複雑性が増すとしてもこれを確保できる場合には問題になりません。

この方法の実際の実装は要件に応じて異なります。通常、主な課題は、この方法を容易に実行するための追加の容量を確保することにあります。

図3.1 Blue-Green デプロイメント



3.2. BLUE-GREEN アップグレードの準備

「インプレースアップグレード」で説明されている方法を使用してマスターと etcd ホストをアップグレードした後に、以下のセクションを参照して残りのノードホストの Blue-Green アップグレードの環境を準備します。

3.2.1. ソフトウェアエンタイトルメントの共有

管理者は Blue-Green デプロイメント間で Red Hat ソフトウェアエンタイトルメントを一時的に共有するか、または Red Hat Satellite などのシステムでインストールコンテンツへのアクセスを提供する必要があります。これは、以前のノードホストからのコンシューマー ID を共有して実行できます。

1. アップグレードされるそれぞれの古いノードホストで、コンシューマー ID であるその **`system identity`** 値を書き留めておいてください。

```
# subscription-manager identity | grep system
system identity: 6699375b-06db-48c4-941e-689efd6ce3aa
```

2. 古いノードホストに置き換わるそれぞれの新規 RHEL 7 または RHEL Atomic Host 7 システムで、直前の手順のそれぞれのコンシューマー ID を使用して登録します。

```
# subscription-manager register --consumerid=6699375b-06db-48c4-
941e-689efd6ce3aa
```

重要

正常なデプロイメントの後に、**subscription-manager clean** で古いホストを登録解除し、環境が非コンプライアンスの状態にならないようにします。

3.2.2. Blue ノードのラベリング

実稼働の現在のノードホストに **blue** または **green** のいずれかのラベルが付けられていることを確認する必要があります。以下の例では、現在の実稼働環境は **blue** となり、新規の環境は **green** になります。

1. クラスターに認識されるノード名の現在の一覧を取得します。

```
$ oc get nodes
```

2. 現在の実稼働環境内にあるマスター以外のノードホスト (コンピュートノード) および専用のインフラストラクチャーノードに、**color=blue** のラベルを付けます。

```
$ oc label node --selector=node-
role.kubernetes.io/compute=true,node-role.kubernetes.io/infra=true
color=blue
```

上記のコマンドでは、**--selector** フラグを使用して、関連のノードラベルでクラスターのサブセットと一致し、照合したすべての項目に **color=blue** のラベルを付けます。

3.2.3. Green ノードの作成およびラベリング

新規ノードホストと同じ数を、既存のクラスターに追加して、置き換えるノードホストに対する Green 環境を新たに作成します。

1. 「既存のクラスターへのホストの追加」に記載の手順を使用して、新規ノードホストを追加します。この手順にある **[new_nodes]** グループで、インベントリーファイルを更新するときにこれらの変数が設定されているようにしてください。

   - ノードが「健全」であるとみなされるまで、ワークロードのスケジューリングを遅らせるには (健全性については後の手順で検証します)、新規ノードホストごとに **openshift_schedulable=false** 変数を設定し、これらが初期の時点でスケジュール対象外となるようにします。

   - 新規ホストが OpenShift Container Platform 3.10 に配置されるので、**openshift_node_group_name** 変数を使用して、ホストエントリーごとにノードグループを設定する必要があります。詳細は、もう一度「ノードグループおよびホストマッピングの定義」を参照してください。

   たとえば、以下の新しいノードホストが既存のインベントリーに追加されます。以前のインベントリーに含まれていた項目はすべて残して置く必要があります。

   例 [new_nodes] ホストグループ

```
[new_nodes]
node4.example.com openshift_node_group_name='node-config-compute'
openshift_schedulable=false
node5.example.com openshift_node_group_name='node-config-compute'
openshift_schedulable=false
node6.example.com openshift_node_group_name='node-config-compute'
```

```
openshift_schedulable=false
infra-node3.example.com openshift_node_group_name='node-config-
infra' openshift_schedulable=false
infra-node4.example.com openshift_node_group_name='node-config-
infra' openshift_schedulable=false
```

2. 新規ノードをデプロイしてから、**oc label node** コマンドを使用して、ノードごとに **color=green** ラベルを適用します。

```
$ oc label node <node_name> color=green
```

3.2.4. Green ノードの検証

新規 Green ノードが健全な状態にあることを確認します。以下のチェックリストを実行します。

1. 新規ノードがクラスター内で検出され、**Ready,SchedulingDisabled** 状態にあることを確認します。

```
$ oc get nodes

NAME                STATUS                        ROLES       AGE
node4.example.com   Ready,SchedulingDisabled      compute     1d
```

2. Green ノードに適切なラベルがあることを確認します。

```
$ oc get nodes --show-labels

NAME                STATUS                        ROLES       AGE
LABELS
node4.example.com   Ready,SchedulingDisabled      compute     1d
beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,color=gree
n,kubernetes.io/hostname=m01.example.com,node-
role.kubernetes.io/compute=true
```

3. クラスターの診断チェックを実行します。

```
$ oc adm diagnostics

[Note] Determining if client configuration exists for client/cluster
diagnostics
Info:  Successfully read a client config file at
'/root/.kube/config'
Info:  Using context for cluster-admin access: 'default/m01-example-
com:8443/system:admin'
[Note] Performing systemd discovery

[Note] Running diagnostic: ConfigContexts[default/m01-example-
com:8443/system:admin]
       Description: Validate client config context is complete and
has connectivity
...
         [Note] Running diagnostic: CheckExternalNetwork
              Description: Check that external network is
accessible within a pod
```

```
       [Note] Running diagnostic: CheckNodeNetwork
              Description: Check that pods in the cluster can
access its own node.

       [Note] Running diagnostic: CheckPodNetwork
              Description: Check pod to pod communication in the
cluster. In case of ovs-subnet network plugin, all pods
should be able to communicate with each other and in case of
multitenant network plugin, pods in non-global projects
should be isolated and pods in global projects should be able to
access any pod in the cluster and vice versa.

       [Note] Running diagnostic: CheckServiceNetwork
              Description: Check pod to service communication in
the cluster. In case of ovs-subnet network plugin, all
pods should be able to communicate with all services and in case of
multitenant network plugin, services in non-global
projects should be isolated and pods in global projects should be
able to access any service in the cluster.
...
```

3.3. GREEN ノードの準備

Pod を Blue 環境から Green に移行するために、必要なコンテナーイメージをプルする必要があります。レジストリーについてのネットワークの待機時間およびロードにより、環境に十分な容量が組み込まれていない場合は遅延が生じる可能性があります。

多くの場合、実行中のシステムへの影響を最小限に抑えるための最良の方法として、新規ノードに到達する新規 Pod デプロイメントをトリガーすることができます。これは、新規イメージストリームをインポートして実行できます。

メジャーリリースの OpenShift Container Platform (および一部の非同期エラータ更新) では、Source-to-Image (S2I) のユーザーのビルダーイメージについての新規イメージストリームを導入しています。インポート時に、「イメージ変更トリガー」で設定されるビルドおよびデプロイメントが自動的に作成されます。

ビルドをトリガーする別の利点として、各種のビルダーイメージ、Pod インフラストラクチャーイメージおよびデプロイヤーなどの多数の補助イメージをすべてのノードホストに効果的に取り込める点があります。Green ノードは **準備完了** (予想される負荷の増加に対応できる状態にある) とみなされると、他のすべてのものが後の手順のノードの退避で移行するため、結果として処理がより迅速になります。

アップグレードプロセスに進む準備ができたら、以下の手順に従って Green ノードを準備します。

1. Green ノードをスケジュール対象 (Schedulable) に設定し、新規 Pod がそれらのノードにのみ到達するようにします。

```
$ oc adm manage-node --schedulable=true --selector=color=green
```

2. Blue ノードを無効にし、それらをスケジュール対象外 (Unschedulable) に設定して新規 Pod がそれらのノードで実行されないようにします。

```
$ oc adm manage-node --schedulable=false --selector=color=blue
```

3. **node-role.kubernetes.io/infra=true** ラベルを使用するように、レジストリーのノード

セレクターとルーターデプロイメント設定を更新します。これにより、新規の Pod が新しいインフラストラクチャーノードに配置されるように、新しいデプロイメントがトリガーされます。

a. **docker-registry** デプロイメント設定を編集します。

```
$ oc edit -n default dc/docker-registry
```

nodeSelector フィールドを以下のように更新して (引用を含めた **"true"** と全く同じ形式を使用)、変更を保存します。

```
      nodeSelector:
        node-role.kubernetes.io/infra: "true"
```

b. **router** デプロイメント設定を編集します。

```
$ oc edit -n default dc/router
```

nodeSelector フィールドを以下のように更新して (引用を含めた **"true"** と全く同じ形式を使用)、変更を保存します。

```
      nodeSelector:
        node-role.kubernetes.io/infra: "true"
```

c. 新規インフラストラクチャーノードで **docker-registry** および **router** Pod が実行中で、Ready の状態であることを確認します。

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

NAME                          READY     STATUS    RESTARTS   AGE
IP                NODE
docker-registry-2-b7xbn       1/1       Running   0          18m
10.128.0.188      infra-node3.example.com
router-2-mvq6p                1/1       Running   0          6m
192.168.122.184   infra-node4.example.com
```

4. デフォルトのイメージストリームとテンプレートを更新します。

5. 最新のイメージをインポートします。このプロセスで多数のビルドがトリガーされる点を理解することが重要です。ただし、ビルドは Green ノードで実行されるため、これが Blue デプロイメントのトラフィックに影響を与えることはありません。

6. クラスター内のすべての namespace (プロジェクト) でのビルドプロセスをモニターするには、以下を実行します。

```
$ oc get events -w --all-namespaces
```

大規模な環境では、ビルドはほとんど停止することがありません。しかしながら、管理上のイメージのインポートによって大きな増減が生じます。

3.4. BLUE ノードの退避および使用停止

大規模デプロイメントでは、退避の調整方法を定めるのに役立つ他のラベルを使用できます。ダウンタイムを防ぐための最も保守的な方法として、一度に 1 つのノードホストを退避する方法があります。

サービスがゾーンの非アフィニティーを使用して複数の Pod で構成されている場合、ゾーン全体を一度に退避できます。使用されるストレージボリュームが新規ゾーンで利用可能であることを確認することは重要です。これについての詳細はクラウドプロバイダーごとに異なる場合があるためです。

OpenShift Container Platform 3.2 以降では、ノードホストの退避はノードサービスの停止時に常にトリガーされます。ノードのラベリングは非常に重要であり、ノードに誤ったラベルが付けられていたり、コマンドが汎用化されたラベルの付いたノードで実行されたりする場合は問題が生じる可能性があります。マスターホストにも **color=blue** のラベルが付けられている場合には注意が必要です。

アップグレードプロセスに進む準備ができたら、以下の手順に従います。

1. 以下のコマンドで、Blue ノードをすべて退避して削除します。

```
$ oc adm manage-node --selector=color=blue --evacuate
$ oc delete node --selector=color=blue
```

2. Blue ノードホストから Pod がなくなり、Blue ノードホストが OpenShift Container Platform から削除されたら、電源をオフにしても問題がありません。安全上の措置として、アップグレードに問題がある場合には、ホストを短時間そのままにしておくと良いでしょう。

3. これらのホストを停止する前に、すべての必要なスクリプトまたはファイルが取得されていることを確認します。指定した期間と容量に問題がないことを確認した後に、これらのホストを削除します。

第4章 オペレーティングシステムの更新

4.1. 目的

メジャーリリース間でのアップグレードや、マイナーリリースのソフトウェアの更新のいずれかによってホストでオペレーティングシステム (OS) を更新すると、それらのマシンで実行されている OpenShift Container Platform ソフトウェアに影響が及びます。とくに、これらの更新は、OpenShift Container Platform で動作する必要のある **iptables** ルールまたは **ovs** フローに影響を与えます。

4.2. ホストでのオペレーティングシステムの更新

以下を使用してホストで OS を安全にアップグレードします。

1. メンテナンスの準備のためにノードをドレイン (解放) します。

```
$ oc adm drain <node_name> --force --delete-local-data --ignore-
daemonsets
```

2. ホストパッケージを更新し、ホストを再起動します。再起動により、ホストが最新バージョンを実行していることを確認できます。つまり、**docker** および OpenShift Container Platform プロセスが再起動されていることになり、これにより他のサービスのすべてのサービスが正しいことを確認するチェックが強制的に実行されます。
ただし、ノードホストを再起動する代わりに、影響を受けるサービスを再起動するか、または **iptables** 状態を保持することができます。どちらのプロセスについても、「OpenShift Container Platform iptables」のトピックで説明されています。**ovs** フロールールは保存される必要はありませんが、OpenShift Container Platform ノードソフトウェアを再起動するとフロールールが固定されます。

3. ホストを再びスケジュール対象 (Schedulable) に設定するには、以下を実行します。

```
$ oc adm uncordon <node_name>
```

第5章 OPENSHIFT のダウングレード

5.1. 概要

OpenShift Container Platform の「アップグレード」後に、極端なケースではあるものの、クラスターを以前のバージョンにダウングレードする必要がある場合があります。以下のセクションでは、OpenShift Container Platform 3.10 から 3.9 へのダウングレードパスなどの、クラスターの各システムでダウングレードを実行するために必要な手順について説明します。

重要

現時点で、これらの手順は OpenShift Container Platform の「RPM ベースのインストール」でのみサポートされており、クラスター全体でダウンタイムが生じることが前提とされています。

5.2. バックアップの確認

1. 「アップグレードプロセス」で使用される Ansible playbook により、**master-config.yaml** ファイルのバックアップが作成されているはずです。このファイルと **scheduler.json** ファイルがマスターに存在しているようにしてください。

```
/etc/origin/master/master-config.yaml.<timestamp>
/etc/origin/master/scheduler.json
```

2. 「自動アップグレードの準備」の手順では、OpenShift Container Platform 3.9 から 3.10 にアップグレードする前に、以下のファイルをバックアップするように指示されています。これらのファイルが利用できることを確認してください。
マスターホスト:

```
/usr/lib/systemd/system/atomic-openshift-master-api.service
/usr/lib/systemd/system/atomic-openshift-master-controllers.service
/etc/sysconfig/atomic-openshift-master-api
/etc/sysconfig/atomic-openshift-master-controllers
```

ノードとマスターホスト:

```
/usr/lib/systemd/system/atomic-openshift-*.service
/etc/origin/node/node-config.yaml
```

etcd ホスト (etcd が共存するマスターを含む):

```
/etc/etcd/etcd.conf
/backup/etcd-xxxxxx/backup.db
```

5.3. クラスターのシャットダウン

1. すべてのマスターおよびノードホストで、以下を実行します。

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

5.4. RPM と静的 POD の削除

***-excluder** パッケージは、インストール時に、エントリーをホストの **/etc/yum.conf** ファイルの exclude ディレクティブに追加します。

1. すべてのマスター、ノードおよび etcd メンバー (専用 etcd クラスターを使用している場合) で、以下のパッケージを削除します。

```
# yum remove atomic-openshift \
    atomic-openshift-clients \
    atomic-openshift-node \
    atomic-openshift-master \
    atomic-openshift-sdn-ovs \
    atomic-openshift-excluder \
    atomic-openshift-docker-excluder \
    atomic-openshift-hyperkube
```

2. パッケージが正常に削除されたことを確認します。

```
# rpm -qa | grep atomic-openshift
```

3. コントロールプレーンホスト (マスターおよび etcd ホスト) で静的な pod 定義を移動します。

```
# mkdir /etc/origin/node/pods-backup
# mv /etc/origin/node/pods/* /etc/origin/node/pods-backup/
```

4. 各ホストを再起動します。

```
# systemctl reboot
```

5.5. DOCKER のダウングレード

OpenShift Container Platform 3.9 も 3.10 も Docker 1.13 が必要ですので、Docker をダウングレードする必要はありません。

5.6. RPM の再インストール

1. OpenShift Container Platform 3.10 のリポジトリーを無効にし、3.9 のリポジトリーを再び有効にします。

```
# subscription-manager repos \
    --disable=rhel-7-server-ose-3.10-rpms \
    --enable=rhel-7-server-ose-3.9-rpms
```

2. 各マスターで、以下のパッケージをインストールします。

```
# yum install atomic-openshift \
    atomic-openshift-clients \
    atomic-openshift-node \
    atomic-openshift-master \
    openvswitch \
    atomic-openshift-sdn-ovs \
```

```
    tuned-profiles-atomic-openshift-node \
    atomic-openshift-excluder \
    atomic-openshift-docker-excluder
```

3. 各ノードで、以下のパッケージをインストールします。

```
# yum install atomic-openshift \
    atomic-openshift-node \
    openvswitch \
    atomic-openshift-sdn-ovs \
    tuned-profiles-atomic-openshift-node \
    atomic-openshift-excluder \
    atomic-openshift-docker-excluder
```

4. 各ホストでパッケージが正常にインストールされたことを確認します。

```
# rpm -qa | grep atomic-openshift
# rpm -q openvswitch
```

5.7. 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
```

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

5.7.1. etcd v3 スナップショットの復元

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** を使用して取得される場合、これには **etcdctl snapshot restore** でチェックされる整合性ハッシュが含まれます。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、復元は **--skip-hash-check** を使用して実行されます。

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. etcd サービスのマスク解除

```
# systemctl unmask etcd
```

2. すべての etcd サービスを停止します。

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

3. 古いデータについては、**etcdctl** が復元手順が実行されるノードでその再作成を実行するため、それらすべてをクリアします。

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

4. **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
```

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

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

6. etcd サービスを起動します。

```
# systemctl start etcd
```

7. エラーメッセージの有無を確認します。

```
# journalctl -fu etcd.service
```

5.7.2. 復元後の etcd ノードの追加

最初のインスタンスを実行後に、複数の etcd サーバーをクラスターに追加できます。

手順

1. **ETCD_NAME** 変数でインスタンスの etcd 名を取得します。

```
# grep ETCD_NAME /etc/etcd/etcd.conf
```

2. etcd がピア通信をリッスンする IP アドレスを取得します。

```
# grep ETCD_INITIAL_ADVERTISE_PEER_URLS /etc/etcd/etcd.conf
```

3. ノードが以前のバージョンで etcd クラスターに含まれていた場合には、以前の etcd データを削除します。

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

4. etcd が正しく実行されている etcd ホストで、新しいメンバーを追加します。

```
# etcdctl3 member add *<name>* \
  --peer-urls="*<advertise_peer_urls>*"
```

このコマンドでは、以下のような変数が出力されます。

```
ETCD_NAME="master2"
ETCD_INITIAL_CLUSTER="master-
0.example.com=https://192.168.55.8:2380"
ETCD_INITIAL_CLUSTER_STATE="existing"
```

5. 新しいホストの **/etc/etcd/etcd.conf** ファイルに、以前のコマンドからの値を追加します。

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

6. クラスターに参加するノードで etcd サービスを起動します。

```
# systemctl start etcd.service
```

7. エラーメッセージの有無を確認します。

```
# journalctl -fu etcd.service
```

8. すべての etcd ノードが追加されるまで、以前の手順を繰り返します。

9. 全ノードを追加したら、クラスターのステータスと、健全性を確認します。

```
# etcdctl3 endpoint health --
endpoints="https://<etcd_host1>:2379,https://<etcd_host2>:2379,https
://<etcd_host3>:2379"
https://master-0.example.com:2379 is healthy: successfully committed
proposal: took = 1.423459ms
https://master-1.example.com:2379 is healthy: successfully committed
proposal: took = 1.767481ms
https://master-2.example.com:2379 is healthy: successfully committed
proposal: took = 1.599694ms

# etcdctl3 endpoint status --
endpoints="https://<etcd_host1>:2379,https://<etcd_host2>:2379,https
://<etcd_host3>:2379"
https://master-0.example.com:2379, 40bef1f6c79b3163, 3.2.5, 28 MB,
true, 9, 2878
https://master-1.example.com:2379, 1ea57201a3ff620a, 3.2.5, 28 MB,
false, 9, 2878
https://master-2.example.com:2379, 59229711e4bc65c8, 3.2.5, 28 MB,
false, 9, 2878
```

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

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

手順

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

```
# cp ${MYBACKUPDIR}/etc/sysconfig/atomic-openshift-master-api
/etc/sysconfig/atomic-openshift-master-api
# cp ${MYBACKUPDIR}/etc/sysconfig/atomic-openshift-master-
controllers /etc/sysconfig/atomic-openshift-master-controllers
# cp ${MYBACKUPDIR}/etc/origin/master/master-config.yaml.<timestamp>
/etc/origin/master/master-config.yaml
# cp ${MYBACKUPDIR}/etc/origin/node/node-config.yaml.<timestamp>
/etc/origin/node/node-config.yaml
# cp ${MYBACKUPDIR}/etc/origin/master/scheduler.json.<timestamp>
/etc/origin/master/scheduler.json
# cp ${MYBACKUPDIR}/usr/lib/systemd/system/atomic-openshift-master-
api.service /usr/lib/systemd/system/atomic-openshift-master-
api.service
# cp ${MYBACKUPDIR}/usr/lib/systemd/system/atomic-openshift-master-
controllers.service /usr/lib/systemd/system/atomic-openshift-master-
controllers.service
# rm /etc/systemd/system/atomic-openshift-node.service
# systemctl daemon-reload
# systemctl enable atomic-openshift-master-api
# systemctl enable atomic-openshift-master-controllers
# systemctl enable atomic-openshift-node
# systemctl start atomic-openshift-master-api
# systemctl start atomic-openshift-master-controllers
# systemctl start atomic-openshift-node
```

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

```
# cp /etc/origin/node/node-config.yaml.<timestamp>
/etc/origin/node/node-config.yaml
# rm /etc/systemd/system/atomic-openshift-node.service
# systemctl daemon-reload
# systemctl enable atomic-openshift-node
# systemctl start atomic-openshift-node
```

5.9. ダウングレードの検証

1. ダウングレードを検証するには、すべてのノードに **Ready** のマークが付けられていることを確認します。

```
# oc get nodes
NAME                      STATUS                     AGE
master.example.com        Ready,SchedulingDisabled   165d
node1.example.com         Ready                      165d
node2.example.com         Ready                      165d
```

2. デプロイされている場合には、レジストリーおよびルーターが正常にダウングレードされたことを確認します。

   a. **v3.9** バージョンの **docker-registry** と **router** のイメージを実行していることを確認します。

```
# oc get -n default dc/docker-registry -o json | grep \"image\"
    "image": "openshift3/ose-docker-registry:v3.9",
# oc get -n default dc/router -o json | grep \"image\"
    "image": "openshift3/ose-haproxy-router:v3.9",
```

   b. **docker-registry** と **router** pod が実行中で、Ready の状態であることを確認します。

```
# oc get pods -n default

NAME                       READY     STATUS    RESTARTS   AGE
docker-registry-2-b7xbn    1/1       Running   0          18m
router-2-mvq6p             1/1       Running   0          6m
```

3. 「診断ツール」をマスターで使用し、共通の問題を検索し、提案される方法を確認します。

```
# oc adm diagnostics
...
[Note] Summary of diagnostics execution:
[Note] Completed with no errors or warnings seen.
```