



OpenShift Container Platform 4.2

アプリケーション

OpenShift Container Platform 4.2 でのアプリケーションの作成および管理

OpenShift Container Platform 4.2 アプリケーション

OpenShift Container Platform 4.2 でのアプリケーションの作成および管理

法律上の通知

Copyright © 2020 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux[®] is the registered trademark of Linus Torvalds in the United States and other countries.

Java[®] is a registered trademark of Oracle and/or its affiliates.

XFS[®] is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL[®] is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js[®] is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack[®] Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

概要

本書では、OpenShift Container Platform で実行されるユーザーによってプロビジョニングされたアプリケーションのインスタンスを作成し、管理する各種の方法について説明します。これには、プロジェクトの使用および Open Service Broker API を使用したアプリケーションのプロビジョニングについての情報が含まれます。

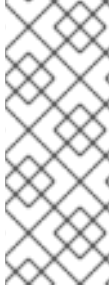
目次

第1章 プロジェクト	3
1.1. プロジェクトの使用	3
1.2. 別のユーザーとしてのプロジェクトの作成	7
1.3. プロジェクト作成の設定	7
第2章 アプリケーションライフサイクル管理	12
2.1. DEVELOPER パースペクティブを使用したアプリケーションの作成	12
2.2. インストールされた OPERATOR からのアプリケーションの作成	15
2.3. CLI を使用したアプリケーションの作成	17
2.4. TOPOLOGY ビューを使用したアプリケーション構成の表示	25
第3章 サービスブローカー	32
3.1. サービスカタログのインストール	32
3.2. テンプレートサービスブローカーのインストール	33
3.3. テンプレートアプリケーションのプロビジョニング	35
3.4. テンプレートサービスブローカーのアンインストール	36
3.5. OPENSIFT ANSIBLE BROKER のインストール	37
3.6. OPENSIFT ANSIBLE BROKER の設定	42
3.7. サービスバンドルのプロビジョニング	45
3.8. OPENSIFT ANSIBLE BROKER のアンインストール	46
第4章 DEPLOYMENT	49
4.1. DEPLOYMENT および DEPLOYMENTCONFIG について	49
4.2. デプロイメントプロセスの管理	55
4.3. DEPLOYMENTCONFIG ストラテジーの使用	62
4.4. ルートベースのデプロイメントストラテジーの使用	69
第5章 クォータ	77
5.1. プロジェクトごとのリソースクォータ	77
5.2. 複数のプロジェクト間のリソースクォータ	89
第6章 アプリケーションの正常性のモニタリング	93
6.1. ヘルスチェックについて	93
6.2. ヘルスチェックの設定	95
第7章 アプリケーションのアイドルリング	99
7.1. アプリケーションのアイドルリング	99
7.2. アプリケーションのアイドルリング解除	99
第8章 リソースを回収するためのオブジェクトのプルーニング	101
8.1. プルーニングの基本操作	101
8.2. グループのプルーニング	101
8.3. デプロイメントのプルーニング	102
8.4. ビルドのプルーニング	102
8.5. イメージのプルーニング	103
8.6. レジストリーのハードプルーニング	109
8.7. CRON ジョブのプルーニング	112

第1章 プロジェクト

1.1. プロジェクトの使用

プロジェクトを使用することにより、あるユーザーコミュニティは、他のコミュニティと切り離された状態で独自のコンテンツを整理し、管理することができます。

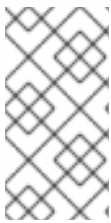


注記

openshift- および **kube-** で始まる名前のプロジェクトはデフォルトプロジェクトです。これらのプロジェクトは、Podとして実行されるクラスターコンポーネントおよび他のインフラストラクチャーコンポーネントをホストします。そのため、OpenShift Container Platform では **oc new-project** コマンドを使用して **openshift-** または **kube-** で始まる名前のプロジェクトを作成することができません。クラスター管理者は、**oc adm new-project** コマンドを使用してこれらのプロジェクトを作成できます。

1.1.1. Web コンソールを使用したプロジェクトの作成

クラスター管理者が許可する場合、新規プロジェクトを作成できます。



注記

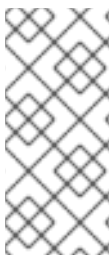
openshift- および **kube-** で始まる名前のプロジェクトは OpenShift Container Platform によって重要 (Critical) と見なされます。そのため、OpenShift Container Platform では、Web コンソールを使用して **openshift-** で始まる名前のプロジェクトを作成することはできません。

手順

1. Home → Projects に移動します。
2. Create Project をクリックします。
3. プロジェクトの詳細を入力します。
4. Create をクリックします。

1.1.2. Web コンソールでの Developer パースペクティブを使用したプロジェクトの作成

OpenShift Container Platform Web コンソールの Developer パースペクティブを使用し、namespace でプロジェクトを作成できます。



注記

openshift- および **kube-** で始まる名前のプロジェクトは、Podとして実行されるクラスターコンポーネントおよび他のインフラストラクチャーコンポーネントをホストします。そのため、OpenShift Container Platform では、CLIを使用して **openshift-** または **kube-** で始まる名前のプロジェクトを作成することはできません。クラスター管理者は、**oc adm new-project** コマンドを使用してこれらのプロジェクトを作成できます。

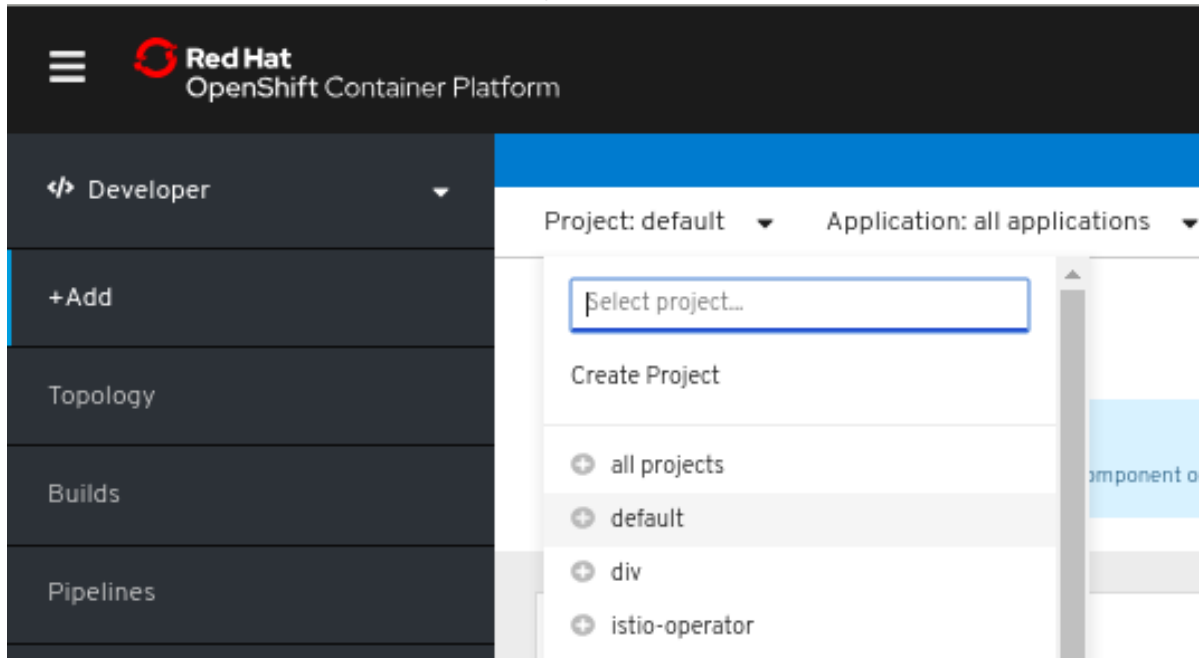
前提条件

- OpenShift Container Platform のプロジェクト、アプリケーション、および他のワークロードを作成するために適切な **ロール** および **パーミッション** があることを確認します。

手順

以下のように、Developer パースペクティブを使用してプロジェクトを作成できます。

1. Add ビューでは、Project ドロップダウンメニューをクリックして、利用可能なすべてのプロジェクトの一覧を表示します。Create Project を選択します。



2. Create Project ダイアログボックスで、Name フィールドに一意の名前を入力します。たとえば、Name フィールドにプロジェクトの名前として **myproject** を入力します。
3. オプション: プロジェクトの Display Name および Description の詳細を追加します。
4. Create をクリックします。

これでプロジェクトが作成され、アプリケーションおよび他のワークロードをこのプロジェクトに追加することができます。

1.1.3. CLI を使用したプロジェクトの作成

クラスター管理者が許可する場合、新規プロジェクトを作成できます。



注記

openshift- および **kube-** で始まる名前のプロジェクトは OpenShift Container Platform によって重要 (Critical) と見なされます。そのため、OpenShift Container Platform では **oc new-project** コマンドを使用して **openshift-** または **kube-** で始まる名前のプロジェクトを作成することができません。クラスター管理者は、**oc adm new-project** コマンドを使用してこれらのプロジェクトを作成できます。

手順

1. 以下を実行します。


```
$ oc new-project <project_name> \  
--description="<description>" --display-name="<display_name>"
```

以下は例になります。

```
$ oc new-project hello-openshift \  
--description="This is an example project" \  
--display-name="Hello OpenShift"
```



注記

作成できるプロジェクトの数は、システム管理者によって制限される場合があります。上限に達すると、新規プロジェクトを作成できるように既存プロジェクトを削除しなければならない場合があります。

1.1.4. Web コンソールを使用したプロジェクトの表示

手順

1. **Home** → **Projects** に移動します。
2. 表示するプロジェクトを選択します。
このページで、**Workloads** ボタンをクリックして、プロジェクトのワークロードを確認します。

1.1.5. CLI を使用したプロジェクトの表示

プロジェクトを表示する際は、認証ポリシーに基づいて、表示アクセスのあるプロジェクトだけを表示できるように制限されます。

手順

1. プロジェクトの一覧を表示するには、以下を実行します。

```
$ oc get projects
```

2. CLI 操作について現在のプロジェクトから別のプロジェクトに切り換えることができます。その後の操作についてはすべて指定のプロジェクトが使用され、プロジェクトスコープのコンテンツの操作が実行されます。

```
$ oc project <project_name>
```

1.1.6. プロジェクトへの追加

手順

1. Web コンソールのナビゲーションメニューの上部にあるコンテキストセクターから **Developer** を選択します。
2. **+Add** をクリックします。
3. ページの上部で、追加するプロジェクトの名前を選択します。

4. プロジェクトに追加する方法をクリックし、ワークフローに従います。

1.1.7. Web コンソールを使用したプロジェクトステータスの確認

手順

1. **Home** → **Projects** に移動します。
2. ステータスを確認するプロジェクトを選択します。

1.1.8. CLI を使用したプロジェクトステータスの確認

手順

1. 以下を実行します。

```
$ oc status
```

このコマンドは、コンポーネントとそれらの各種の関係を含む現在のプロジェクトの概要を示します。

1.1.9. Web コンソールを使用したプロジェクトの削除


OpenShift Container Platform Web コンソールを使用してプロジェクトを削除できます。



注記

プロジェクトを削除するパーミッションがない場合は、Delete Project オプションが選択できなくなります。

手順

1. **Home** → **Projects** に移動します。
2. プロジェクトの一覧から削除するプロジェクトを見つけます。
3. プロジェクト一覧の右側にある Options メニューから **Delete Project** を選択します。  をクリックします。
4. **Delete Project** ペインが開いたら、フィールドから削除するプロジェクトの名前を入力します。
5. **Delete** をクリックします。

1.1.10. CLI を使用したプロジェクトの削除

プロジェクトを削除する際に、サーバーはプロジェクトのステータスを **Active** から **Terminating** に更新します。次に、サーバーは **Terminating** 状態のプロジェクトからすべてのコンテンツをクリアしてから、最終的にプロジェクトを削除します。プロジェクトのステータスが **Terminating** の場合、新規のコンテンツをプロジェクトに追加することはできません。プロジェクトは CLI または Web コンソールから削除できます。

手順

1. 以下を実行します。

```
$ oc delete project <project_name>
```

1.2. 別のユーザーとしてのプロジェクトの作成

権限の借用機能により、別のユーザーとしてプロジェクトを作成することができます。

1.2.1. API の権限借用

OpenShift Container Platform API への要求を、別のユーザーから発信されているかのように設定できます。詳細は、Kubernetes ドキュメントの「[User impersonation](#)」を参照してください。

1.2.2. プロジェクト作成時のユーザー権限の借用

プロジェクト要求を作成する際に別のユーザーの権限を借用できます。**system:authenticated:oauth** はプロジェクト要求を作成できる唯一のブートストラップグループであるため、そのグループの権限を借用する必要があります。

手順

- 別のユーザーの代わりにプロジェクト要求を作成するには、以下を実行します。

```
$ oc new-project <project> --as=<user> \  
--as-group=system:authenticated --as-group=system:authenticated:oauth
```

1.3. プロジェクト作成の設定

OpenShift Container Platform では、**プロジェクト** は関連するオブジェクトをグループ分けし、分離するために使用されます。Web コンソールまたは **oc new-project** コマンドを使用して新規プロジェクトの作成要求が実行されると、OpenShift Container Platform のエンドポイントは、カスタマイズ可能なテンプレートに応じてプロジェクトをプロビジョニングするために使用されます。

クラスター管理者は、開発者やサービスアカウントが独自のプロジェクトを作成し、プロジェクトの **セルフプロビジョニング** を実行することを許可し、その方法を設定できます。

1.3.1. プロジェクト作成について

OpenShift Container Platform API サーバーは、クラスターのプロジェクト設定リソースの **projectRequestTemplate** パラメーターで識別されるプロジェクトテンプレートに基づいて新規プロジェクトを自動的にプロビジョニングします。パラメーターが定義されない場合、API サーバーは要求される名前でプロジェクトを作成するデフォルトテンプレートを作成し、要求するユーザーをプロジェクトの **admin** (管理者) ロールに割り当てます。

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

表1.1 デフォルトのプロジェクトテンプレートパラメーター

パラメーター	説明
PROJECT_NAME	プロジェクトの名前。必須。
PROJECT_DISPLAYNAME	プロジェクトの表示名。空にできます。
PROJECT_DESCRIPTION	プロジェクトの説明。空にできます。
PROJECT_ADMIN_USER	管理ユーザーのユーザー名。
PROJECT_REQUESTING_USER	要求するユーザーのユーザー名。

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

1.3.2. 新規プロジェクトのテンプレートの変更

クラスター管理者は、デフォルトのプロジェクトテンプレートを変更し、新規プロジェクトをカスタム要件に基づいて作成することができます。

独自のカスタムプロジェクトテンプレートを作成するには、以下を実行します。

手順

1. **cluster-admin** 権限を持つユーザーとしてのログイン。
2. デフォルトのプロジェクトテンプレートを生成します。

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

3. オブジェクトを追加するか、または既存オブジェクトを変更することにより、テキストエディターで生成される **template.yaml** ファイルを変更します。
4. プロジェクトテンプレートは、**openshift-config** namespace に作成される必要があります。変更したテンプレートを読み込みます。

```
$ oc create -f template.yaml -n openshift-config
```

5. Web コンソールまたは CLI を使用し、プロジェクト設定リソースを編集します。
 - Web コンソールの使用
 - i. **Administration** → **Cluster Settings** ページに移動します。
 - ii. **Global Configuration** をクリックし、すべての設定リソースを表示します。
 - iii. **Project** のエントリーを見つけ、**Edit YAML** をクリックします。
 - CLI の使用
 - i. **project.config.openshift.io/cluster** リソースを編集します。

```
$ oc edit project.config.openshift.io/cluster
```

6. **spec** セクションを、**projectRequestTemplate** および **name** パラメーターを組み込むように更新し、アップロードされたプロジェクトテンプレートの名前を設定します。デフォルト名は **project-request** です。

カスタムプロジェクトテンプレートを含むプロジェクト設定リソース

```
apiVersion: config.openshift.io/v1
kind: Project
metadata:
  ...
spec:
  projectRequestTemplate:
    name: <template_name>
```

7. 変更を保存した後、変更が正常に適用されたことを確認するために、新しいプロジェクトを作成します。

1.3.3. プロジェクトのセルフプロビジョニングの無効化

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

手順

1. **cluster-admin** 権限を持つユーザーとしてのログイン。
2. 以下のコマンドを実行して、**self-provisioners** クラスタロールバインディングの使用を確認します。

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

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

self-provisioners セクションのサブジェクトを確認します。

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

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

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

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

4. ロールへの自動更新を防ぐには、**self-provisioners** クラスターロールバインディングを編集します。自動更新により、クラスターロールがデフォルトの状態にリセットされます。

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

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

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

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

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

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

5. 認証されたユーザーとしてログインし、プロジェクトのセルフプロビジョニングを実行できないことを確認します。

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

```
Error from server (Forbidden): You may not request a new project via this API.
```

組織に固有のより有用な説明を提供できるようこのプロジェクト要求メッセージをカスタマイズすることを検討します。

1.3.4. プロジェクト要求メッセージのカスタマイズ

プロジェクトのセルフプロビジョニングを実行できない開発者またはサービスアカウントが Web コンソールまたは CLI を使用してプロジェクト作成要求を行う場合、以下のエラーメッセージがデフォルトで返されます。

```
You may not request a new project via this API.
```

クラスター管理者はこのメッセージをカスタマイズできます。これを、組織に固有の新規プロジェクトの要求方法の情報を含むように更新することを検討します。以下は例になります。

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

プロジェクト要求メッセージをカスタマイズするには、以下を実行します。

手順

1. Web コンソールまたは CLI を使用し、プロジェクト設定リソースを編集します。
 - Web コンソールの使用
 - i. **Administration** → **Cluster Settings** ページに移動します。
 - ii. **Global Configuration** をクリックし、すべての設定リソースを表示します。
 - iii. **Project** のエントリーを見つけ、**Edit YAML** をクリックします。
 - CLI の使用
 - i. **cluster-admin** 権限を持つユーザーとしてのログイン。
 - ii. **project.config.openshift.io/cluster** リソースを編集します。

```
$ oc edit project.config.openshift.io/cluster
```

2. **spec** セクションを、**projectRequestMessage** パラメーターを含むように更新し、値をカスタムメッセージに設定します。

カスタムプロジェクト要求メッセージを含むプロジェクト設定リソース

```
apiVersion: config.openshift.io/v1
kind: Project
metadata:
  ...
spec:
  projectRequestMessage: <message_string>
```

以下は例になります。

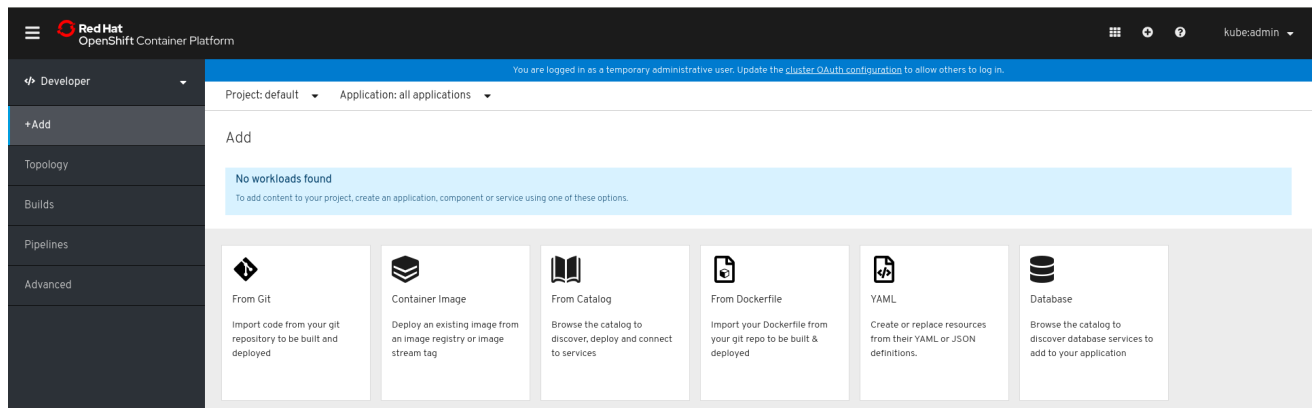
```
apiVersion: config.openshift.io/v1
kind: Project
metadata:
  ...
spec:
  projectRequestMessage: To request a project, contact your system administrator at
  projectname@example.com.
```

3. 変更を保存した後に、プロジェクトをセルフプロビジョニングできない開発者またはサービスアカウントとして新規プロジェクトの作成を試行し、変更が正常に適用されていることを確認します。

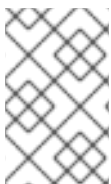
第2章 アプリケーションライフサイクル管理

2.1. DEVELOPER パースペクティブを使用したアプリケーションの作成

Web コンソールの **Developer** パースペクティブでは、**Add** ビューからアプリケーションおよび関連サービスを作成し、それらを OpenShift Container Platform にデプロイするための以下のオプションが提供されます。



- **From Git** このオプションを使用して、Git リポジトリの既存のコードベースをインポートし、OpenShift Container Platform でアプリケーションを作成し、ビルドし、デプロイします。
- **Container Image**: イメージストリームまたはレジストリーからの既存イメージを使用し、これを OpenShift Container Platform にデプロイします。
- **From Catalog: Developer Catalog** で、イメージビルダーに必要なアプリケーション、サービス、またはソースを選択し、これをプロジェクトに追加します。
- **From Dockerfile**: Git リポジトリから dockerfile をインポートし、アプリケーションをビルドし、デプロイします。
- **YAML**: エディターを使用して YAML または JSON 定義を追加し、リソースを作成し、変更します。
- **Database**: **Developer Catalog** を参照して、必要なデータベースサービスを選択し、これをアプリケーションに追加します。



注記

上記のオプションの Serverless セクションは、**Serverless Operator** がクラスターにインストールされている場合にのみ表示されます。OpenShift Serverless はテクノロジーレビュー機能です。

前提条件

Developer パースペクティブを使用してアプリケーションを作成するには、以下を確認してください。

- **Web コンソールにログイン**している。
- **Developer** パースペクティブにいる。
- OpenShift Container Platform でアプリケーションおよび他のワークロードを作成するための適切なプロジェクト内の**ロールおよびパーミッション**がある。

2.1.1. Git のコードベースのインポートおよびアプリケーションの作成

以下の手順では、**Developer** パースペクティブの **Git** からのインポートオプションを使用してアプリケーションを作成します。

以下のように GitHub で既存のコードベースを使用して、OpenShift Container Platform でアプリケーションを作成し、ビルドし、デプロイします。

手順

1. **Add** ビューで **From Git** をクリックし、**Import from Git** フォームを表示します。

Import from git

Git

Git Repo URL *

> Show Advanced Git Options

Builder

Builder Image *

Perl	PHP	Nginx	Modern Webapp	Httpd	.NET Core	Go	Ruby	Python	Java	Node.js

General

Application

Create Application

Select an application for your grouping or Unassigned to not use an application grouping

Application Name

A unique name given to the application grouping to label your resources.

Name *

A unique name given to the component that will be used to name associated resources.

Advanced Options

Create a route to the application
Exposes your application at a public URL

Click on the names to access advanced options for [Routing](#), [Build Configuration](#), [Deployment Configuration](#), [Scaling](#), [Resource Limits](#) and [Labels](#).

Create Cancel

2. **Git** セクションで、アプリケーションの作成に使用するコードベースの Git リポジトリ URL を入力します。たとえば、このサンプル nodejs アプリケーションの URL **https://github.com/sclorg/nodejs-ex** を入力します。
3. オプション: **Show Advanced Git Options** をクリックし、以下のような詳細を追加できます。
 - **Git Reference:** アプリケーションのビルドに使用する特定のブランチ、タグ、またはコミットのコードを参照します。
 - **Context Dir:** アプリケーションのビルドに使用するアプリケーションのソースコードのサブディレクトリを指定します。
 - **Source Secret** プライベートリポジトリからソースコードをプルするための認証情報で **Secret Name** を作成します。
4. **Builder** セクションで、必要なビルダーイメージを選択して、ビルダーイメージの詳細を確認します。必要に応じて、**Builder Image Version** のドロップダウンリストを使用してバージョンを変更できます。たとえば、**Node.js** ビルダーイメージを選択します。
5. **General** セクションで、以下を実行します。

- a. **Application Name** フィールドで、アプリケーションを分類するために一意の名前 (**myapp** など) を入力します。アプリケーション名が namespace で一意であることを確認します。
- b. **Name** フィールドで、このアプリケーション用に作成されたリソースが自動的に設定されることを確認します。



注記

リソース名は namespace で一意である必要があります。エラーが出る場合はリソース名を変更します。

6. **Serverless** セクションで、**Enable scaling to zero when idle** を選択し、Pod が自動的にゼロにスケールし、アイドル時のリソース消費を防止するサーバーレスアプリケーションを作成します。



注記

Serverless セクションは、**Serverless Operator** がクラスターにインストールされている場合にのみ、**Import from git** フォームに表示されます。詳細は、OpenShift Serverless のインストールについてのドキュメントを参照してください。

7. **Advanced Options** セクションでは、**Create a route to the application** がデフォルトで選択されるため、公開されている URL を使用してアプリケーションにアクセスできます。アプリケーションをパブリックルートに公開したくない場合は、チェックボックスをクリアできます。
8. オプション: 以下の高度なオプションを使用してアプリケーションをさらにカスタマイズできません。

ルーティング

Routing リンクをクリックして、以下を実行します。

- ルートのホスト名をカスタマイズします。
- ルーターが監視するパスを指定します。
- ドロップダウンリストから、トラフィックのターゲットポートを選択します。
- **Secure Route** チェックボックスを選択してルートを保護します。必要な TLS 終端タイプを選択し、各ドロップダウンリストから非セキュアなトラフィックについてのポリシーを設定します。
サーバーレスアプリケーションの場合、Knative Service が上記のすべてのルーティングオプションを管理します。ただし、必要に応じて、トラフィックのターゲットポートをカスタマイズできます。ターゲットポートが指定されていない場合、デフォルトポートの **8080** が使用されます。

ビルドおよびデプロイメント設定

Build Configuration および **Deployment Configuration** リンクをクリックして、各設定オプションを表示します。オプションの一部はデフォルトで選択されています。必要なトリガーおよび環境変数を追加して、オプションをさらにカスタマイズできます。サーバーレスアプリケーションの場合、**Deployment Configuration** オプションは表示されません。これは、Knative 設定リソースが DeploymentConfig の代わりにデプロイメントの状態を維持するためです。

スケールリング

Scaling リンクをクリックして、最初にデプロイするアプリケーションの Pod 数またはインスタンス数を定義します。

サーバーレスアプリケーションの場合、以下を実行できます。

- Autoscaler で設定できる Pod 数の上限および下限を設定します。下限が指定されない場合は、デフォルトでゼロに設定されます。
- 指定された時点でのアプリケーションインスタンスごとに必要な同時要求数のソフト制限を定義します。自動スケーリングの推奨設定です。指定されていない場合は、クラスター設定で指定した値を使用します。
- 指定された時点でのアプリケーションインスタンスごとに許可される同時要求数のハード制限を定義します。これは、リビジョンテンプレートで設定されます。指定されていない場合、デフォルトでクラスター設定で指定された値に設定されます。

リソースの制限

Resource Limit リンクをクリックして、コンテナが実行時に保証または使用が許可されている CPU および メモリー リソースの量を設定します。

ラベル

Labels リンクをクリックして、カスタムラベルをアプリケーションに追加します。

9. **Create** をクリックして、アプリケーションを作成し、**Topology** ビューでビルドのステータスを確認します。

2.2. インストールされた OPERATOR からのアプリケーションの作成

Operator は、Kubernetes アプリケーションをパッケージ化し、デプロイし、管理する方法です。クラスター管理者によってインストールされる Operator を使用して、アプリケーションを OpenShift Container Platform で作成できます。

以下では、開発者を対象に、OpenShift Container Platform Web コンソールを使用して、インストールされた Operator からアプリケーションを作成する例を示します。

追加リソース

- Operator の仕組みおよび Operator Lifecycle Manager の OpenShift Container Platform への統合方法に関する詳細は、『[Operator](#)』ガイドを参照してください。

2.2.1. Operator を使用した etcd クラスターの作成

この手順では、Operator Lifecycle Manager (OLM) で管理される etcd Operator を使用した新規 etcd クラスターの作成について説明します。

前提条件

- OpenShift Container Platform 4.2 クラスターへのアクセス
- 管理者によってクラスターにすでにインストールされている etcd Operator

手順

1. この手順を実行するために OpenShift Container Platform Web コンソールで新規プロジェクトを作成します。この例では、**my-etcd** というプロジェクトを使用します。

2. **Operators → Installed Operators** ページに移動します。クラスター管理者によってクラスターにインストールされ、使用可能にされた Operator が ClusterServiceVersion (CSV) の一覧としてここに表示されます。CSV は Operator によって提供されるソフトウェアを起動し、管理するために使用されます。

ヒント

以下を使用して、CLI でこの一覧を取得できます。

```
$ oc get csv
```

3. **Installed Operators** ページで、**Copied** をクリックしてから、**etcd Operator** をクリックして詳細情報および選択可能なアクションを表示します。

図2.1 etcd Operator の概要

The screenshot displays the OpenShift console interface for the etcd Operator. At the top, there's a header with the etcd logo, the text 'etcd 0.9.2 provided by CoreOS, Inc', and an 'Actions' dropdown menu. Below this is a navigation bar with tabs: 'Overview' (selected), 'YAML', 'Events', 'All Instances', 'etcd Cluster', 'etcd Backup', and 'etcd Restore'. The main content area is split into two columns. The left column lists metadata: 'PROVIDER: CoreOS, Inc', 'CREATED AT: Feb 4, 3:10 pm', 'LINKS: Blog, https://coreos.com/etcd', 'Documentation: https://coreos.com/operator/s/etcd/docs/latest/', 'etcd Operator Source Code: https://github.com/coreos/etcd-operator', and 'MAINTAINERS: CoreOS, Inc, support@coreos.com'. The right column is titled 'Provided APIs' and features three API cards. Each card has a circular icon (EC for etcd Cluster, EB for etcd Backup, ER for etcd Restore), a title, a description, and a 'Create New' button. The 'etcd Cluster' card describes it as representing a cluster of etcd nodes. The 'etcd Backup' card describes it as representing the intent to backup an etcd cluster. The 'etcd Restore' card describes it as representing the intent to restore an etcd cluster from a backup. Below the API cards is a 'Description' section for etcd, explaining it as a distributed key value store for storing data across a cluster of machines, noting it's open-source and available on GitHub.

Provided APIs に表示されているように、この Operator は 3 つの新規リソースタイプを利用可能にします。これには、**etcd クラスター (EtcdCluster リソース)** のタイプが含まれます。これらのオブジェクトは、**Deployments** または **ReplicaSets** などの組み込み済みのネイティブ Kubernetes オブジェクトと同様に機能しますが、これらには etcd を管理するための固有のロジックが含まれます。

4. 新規 etcd クラスターを作成します。
 - a. **etcd Cluster** API ボックスで、**Create New** をクリックします。
 - b. 次の画面では、クラスターのサイズなど **EtcdCluster** オブジェクトのテンプレートを起動する最小条件への変更を加えることができます。ここでは **Create** をクリックして確定します。これにより、Operator がトリガーされ、Pod、サービス、および新規 etcd クラスターの他のコンポーネントが起動します。

5. **Resources** タブをクリックして、プロジェクトに Operator によって自動的に作成され、設定された数多くのリソースが含まれることを確認します。

図2.2 etcd Operator リソース

etcdoperator.v0.9.2 > EtcdCluster Details

EC example

Actions ▾

Overview YAML **Resources**

Filter Resources by name...

2 Service

3 Pod

Select All Filters

5 Items

NAME ↑	TYPE	STATUS	CREATED
S example	Service	Created	🕒 3 minutes ago
S example-client	Service	Created	🕒 3 minutes ago
P example-dccdn267hl	Pod	Running	🕒 2 minutes ago
P example-g2shm4cz4l	Pod	Running	🕒 2 minutes ago
P example-sgm2hcktcn	Pod	Running	🕒 3 minutes ago

Kubernetes サービスが作成され、プロジェクトの他の Pod からデータベースにアクセスできることを確認します。

6. 所定プロジェクトで **edit** ロールを持つすべてのユーザーは、クラウドサービスのようにセルフサービス方式でプロジェクトにすでに作成されている Operator によって管理されるアプリケーションのインスタンス (この例では etcd クラスター) を作成し、管理し、削除することができます。この機能を持つ追加のユーザーを有効にする必要がある場合、プロジェクト管理者は以下のコマンドを使用してこのロールを追加できます。

```
$ oc policy add-role-to-user edit <user> -n <target_project>
```

これで、etcd クラスターは Pod が正常でなくなったり、クラスターのノード間で移行する際の障害に対応し、データのリバランスを行います。最も重要な点として、適切なアクセスを持つクラスター管理者または開発者は独自のアプリケーションでデータベースを簡単に使用できるようになります。

2.3. CLI を使用したアプリケーションの作成

OpenShift Container Platform CLI を使用して、ソースまたはバイナリーコード、イメージおよびテンプレートを含むコンポーネントから OpenShift Container Platform アプリケーションを作成できます。

new-app で作成したオブジェクトのセットは、ソースリポジトリ、イメージまたはテンプレートなどのインプットとして渡されるアーティファクトによって異なります。

2.3.1. ソースコードからのアプリケーションの作成

new-app コマンドを使用して、ローカルまたはリモート Git リポジトリのソースコードからアプリケーションを作成できます。

new-app コマンドは、ビルド設定を作成し、これはソースコードから新規のアプリケーションイメージを作成します。**new-app** コマンドは通常、デプロイメント設定を作成して新規のイメージをデプロイするほか、サービスを作成してイメージを実行するデプロイメントへの負荷分散したアクセスを提供します。

OpenShift Container Platform は、**Pipeline** または **Source** ビルドストラテジーのいずれを使用すべきかを自動的に検出します。また、**Source** ビルドの場合は、適切な言語のビルダーイメージを検出します。

2.3.1.1. ローカル

ローカルディレクトリーの Git リポジトリを使用してアプリケーションを作成するには、以下を実行します。

```
$ oc new-app /<path to source code>
```



注記

ローカル Git リポジトリを使用する場合には、リポジトリで OpenShift Container Platform クラスターがアクセス可能な URL を参照する **origin** という名前のリモートリポジトリが必要です。認識されているリモートがない場合は、**new-app** コマンドを実行してバイナリービルドを作成します。

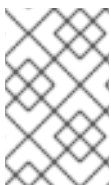
2.3.1.2. リモート

リモート Git リポジトリを使用してアプリケーションを作成するには、以下を実行します。

```
$ oc new-app https://github.com/sclorg/cakephp-ex
```

プライベートのリモート Git リポジトリを使用してアプリケーションを作成するには、以下を実行します。

```
$ oc new-app https://github.com/youruser/yourprivaterepo --source-secret=yoursecret
```



注記

プライベートリモート Git リポジトリを使用する場合には、**--source-secret** フラグを使用して、既存のソースクローンのシークレットを指定できます。このシークレットは、**BuildConfig** に挿入され、リポジトリにアクセスできるようになります。

--context-dir フラグを指定することで、ソースコードリポジトリのサブディレクトリーを使用できます。リモート Git リポジトリおよびコンテキストサブディレクトリーを使用してアプリケーションを作成する場合は、以下を実行します。

```
$ oc new-app https://github.com/sclorg/s2i-ruby-container.git \
  --context-dir=2.0/test/puma-test-app
```

また、リモート URL を指定する場合は、以下のように URL の最後に **#<branch_name>** を追加することで、使用する Git ブランチを指定できます。

```
$ oc new-app https://github.com/openshift/ruby-hello-world.git#beta4
```

2.3.1.3. ビルドストラテジーの検出

新規アプリケーションの作成時に **Jenkinsfile** がソースリポジトリのルート または指定されたコンテキストディレクトリに存在する場合には、OpenShift Container Platform は Pipeline ビルドストラテジーを生成します。

それ以外の場合は、ソースビルドストラテジーが生成されます。

ビルドストラテジーを上書きするには、**--strategy** フラグを **pipeline** または **source**のいずれかに設定します。

```
$ oc new-app /home/user/code/myapp --strategy=docker
```



注記

oc コマンドを使用するには、ビルドソースを含むファイルがリモートの git リポジトリで利用可能である必要があります。すべてのソースビルドには、**git remote -v** を使用する必要があります。

2.3.1.4. 言語の検出

Source ビルドストラテジーを使用する場合には、**new-app** はリポジトリのルート または指定したコンテキストディレクトリに特定のファイルが存在するかどうかで、使用する言語ビルダーを判別しようとします。

表2.1 new-app が検出する言語

言語	ファイル
dotnet	project.json、*.csproj
jee	pom.xml
nodejs	app.json、package.json
perl	cpanfile、index.pl
php	composer.json、index.php
python	requirements.txt、setup.py
ruby	Gemfile、Rakefile、config.ru
scala	build.sbt
golang	Godeps、main.go

言語の検出後、**new-app** は OpenShift Container Platform サーバーで、検出言語と一致する **supports**

アノテーションを持つイメージストリームタグを検索するか、または検出された言語の名前に一致するイメージストリームを検索します。一致するものが見つからない場合には、**new-app** は [Docker Hub レジストリー](#) で名前をベースにした検出言語と一致するイメージの検索を行います。

~をセパレーターとして使用し、イメージ(イメージストリームまたはコンテナの仕様)とリポジトリを指定して、特定のソースリポジトリにビルダーが使用するイメージを上書きすることができます。この方法を使用すると、ビルドストラテジーの検出および言語の検出は実行されない点に留意してください。

たとえば、リモートリポジトリのソースを使用して **myproject/my-ruby** イメージストリームを作成する場合は、以下を実行します。

```
$ oc new-app myproject/my-ruby~https://github.com/openshift/ruby-hello-world.git
```

ローカルリポジトリのソースを使用して **openshift/ruby-20-centos7:latest** コンテナのイメージストリームを作成するには、以下を実行します。

```
$ oc new-app openshift/ruby-20-centos7:latest~/home/user/code/my-ruby-app
```

注記

言語の検出では、リポジトリのクローンを作成し、検査できるように Git クライアントをローカルにインストールする必要があります。Git が使用できない場合、**<image>~<repository>** 構文を指定し、リポジトリで使用するビルダーイメージを指定して言語の検出手順を回避することができます。

-i <image> <repository> 呼び出しでは、アーティファクトのタイプを判別するために **new-app** が **repository** のクローンを試行する必要があります。そのため、これは Git が利用できない場合には失敗します。

-i <image> --code <repository> 呼び出しでは、**image** がソースコードのビルダーとして使用されるか、またはデータベースイメージの場合のように別個にデプロイされる必要があるかどうかを判別するために、**new-app** が **repository** のクローンを作成する必要があります。

2.3.2. イメージからアプリケーションを作成する方法

既存のイメージからアプリケーションのデプロイが可能です。イメージは、OpenShift Container Platform サーバー内のイメージストリーム、指定したレジストリー内のイメージ、またはローカルの Docker サーバー内のイメージから取得できます。

new-app コマンドは、渡された引数に指定されたイメージの種類を判断しようとします。ただし、イメージが、**--docker-image** 引数を使用したコンテナイメージなのか、または **-i|--image** 引数を使用したイメージストリームなのかを、**new-app** に明示的に指示できます。

注記

ローカル Docker リポジトリからイメージを指定した場合、同じイメージが OpenShift Container Platform のクラスターノードでも利用できることを確認する必要があります。

2.3.2.1. DockerHub MySQL イメージ

たとえば、DockerHub MySQL イメージからアプリケーションを作成するには、以下を実行します。


```
$ oc new-app mysql
```

2.3.2.2. プライベートレジストリーのイメージ

プライベートのレジストリーのイメージを使用してアプリケーションを作成し、コンテナイメージの仕様全体を以下のように指定します。

```
$ oc new-app myregistry:5000/example/myimage
```

2.3.2.3. 既存のイメージストリームおよびオプションのイメージストリームタグ

既存のイメージストリームおよびオプションのイメージストリームタグでアプリケーションを作成します。

```
$ oc new-app my-stream:v1
```

2.3.3. テンプレートからのアプリケーションの作成

テンプレート名を引数として指定することで、事前に保存したテンプレートまたはテンプレートファイルからアプリケーションを作成することができます。たとえば、サンプルアプリケーションテンプレートを保存し、これを利用してアプリケーションを作成できます。

保存したテンプレートからアプリケーションを作成します。以下は例になります。

```
$ oc create -f examples/sample-app/application-template-stibuild.json
$ oc new-app ruby-helloworld-sample
```

事前に OpenShift Container Platform に保存することなく、ローカルファイルシステムでテンプレートを直接使用するには、**-f|--file** 引数を使用します。以下は例になります。

```
$ oc new-app -f examples/sample-app/application-template-stibuild.json
```

2.3.3.1. テンプレートパラメーター

テンプレートをベースとするアプリケーションを作成する場合、以下の **-p|--param** 引数を使用してテンプレートで定義したパラメーター値を設定します。

```
$ oc new-app ruby-helloworld-sample \
  -p ADMIN_USERNAME=admin -p ADMIN_PASSWORD=mypassword
```

パラメーターをファイルに保存しておいて、**--param-file** を指定して、テンプレートをインスタンス化する時にこのファイルを使用することができます。標準入力からパラメーターを読み込む場合は、以下のように **--param-file=-** を使用します。

```
$ cat helloworld.params
ADMIN_USERNAME=admin
ADMIN_PASSWORD=mypassword
$ oc new-app ruby-helloworld-sample --param-file=helloworld.params
$ cat helloworld.params | oc new-app ruby-helloworld-sample --param-file=-
```

2.3.4. アプリケーション作成の変更

new-app コマンドは、OpenShift Container Platform オブジェクトを生成します。このオブジェクトにより、作成されるアプリケーションがビルドされ、デプロイされ、実行されます。通常、これらのオブジェクトは現在のプロジェクトに作成され、これらのオブジェクトには入力ソースリポジトリまたはインプットイメージから派生する名前が割り当てられます。ただし、**new-app** でこの動作を変更することができます。

表2.2 **new-app** 出力オブジェクト

オブジェクト	説明
BuildConfig	BuildConfig は、コマンドラインで指定された各ソースリポジトリに作成されます。 BuildConfig は使用するストラテジー、ソースのロケーション、およびビルドの出力ロケーションを指定します。
ImageStreams	BuildConfig では、通常 2 つの ImageStreams が作成されます。1 つ目は、インプットイメージを表します。 Source ビルドの場合、これはビルダーイメージです。 Docker ビルドでは、これは FROM イメージです。2 つ目は、アウトプットイメージを表します。コンテナイメージが new-app にインプットとして指定された場合、このイメージに対してもイメージストリームが作成されます。
DeploymentConfig	DeploymentConfig は、ビルドの出力または指定されたイメージのいずれかをデプロイするために作成されます。 new-app コマンドは、結果として生成される DeploymentConfig に含まれるコンテナに指定されるすべての Docker ボリュームに emptyDir ボリュームを作成します。
Service	new-app コマンドは、インプットイメージで公開ポートを検出しようと試みます。公開されたポートで数値が最も低いものを使用して、そのポートを公開するサービスを生成します。 new-app 完了後に別のポートを公開するには、単に oc expose コマンドを使用し、追加のサービスを生成するだけです。
その他	テンプレートのインスタンスを作成する際に、他のオブジェクトをテンプレートに基づいて生成できます。

2.3.4.1. 環境変数の指定

テンプレート、ソースまたはイメージからアプリケーションを生成する場合、**-e|--env** 引数を使用し、ランタイムに環境変数をアプリケーションコンテナに渡すことができます。

```
$ oc new-app openshift/postgresql-92-centos7 \
  -e POSTGRESQL_USER=user \
  -e POSTGRESQL_DATABASE=db \
  -e POSTGRESQL_PASSWORD=password
```

変数は、**--env-file** 引数を使用してファイルから読み取ることもできます。

```
$ cat postgresql.env
POSTGRESQL_USER=user
POSTGRESQL_DATABASE=db
POSTGRESQL_PASSWORD=password
$ oc new-app openshift/postgresql-92-centos7 --env-file=postgresql.env
```

さらに **--env-file=** を使用することで、標準入力で環境変数を指定することもできます。

```
$ cat postgresql.env | oc new-app openshift/postgresql-92-centos7 --env-file=
```



注記

-e|--env または **--env-file** 引数で渡される環境変数では、**new-app** 処理の一環として作成される **BuildConfig** オブジェクトは更新されません。

2.3.4.2. ビルド環境変数の指定

テンプレート、ソースまたはイメージからアプリケーションを生成する場合、**--build-env** 引数を使用し、ランタイムに環境変数をビルドコンテナに渡すことができます。

```
$ oc new-app openshift/ruby-23-centos7 \
  --build-env HTTP_PROXY=http://myproxy.net:1337/ \
  --build-env GEM_HOME=~/.gem
```

変数は、**--build-env-file** 引数を使用してファイルから読み取ることもできます。

```
$ cat ruby.env
HTTP_PROXY=http://myproxy.net:1337/
GEM_HOME=~/.gem
$ oc new-app openshift/ruby-23-centos7 --build-env-file=ruby.env
```

さらに **--build-env-file=** を使用して、環境変数を標準入力で指定することもできます。

```
$ cat ruby.env | oc new-app openshift/ruby-23-centos7 --build-env-file=
```

2.3.4.3. ラベルの指定

ソース、イメージ、またはテンプレートからアプリケーションを生成する場合、**-l|--label** 引数を使用し、作成されたオブジェクトにラベルを追加できます。ラベルを使用すると、アプリケーションに関連するオブジェクトを一括で選択、設定、削除することが簡単になります。

```
$ oc new-app https://github.com/openshift/ruby-hello-world -l name=hello-world
```

2.3.4.4. 作成前の出力の表示

new-app コマンドの実行に関するドライランを確認するには、**yaml** または **json** の値と共に **-o|--output** 引数を使用できます。次にこの出力を使用して、作成されるオブジェクトのプレビューまたは編集可能なファイルへのリダイレクトを実行できます。問題がなければ、**oc create** を使用して OpenShift Container Platform オブジェクトを作成できます。

new-app アーティファクトをファイルに出力するには、これらを編集し、作成します。

```
$ oc new-app https://github.com/openshift/ruby-hello-world \
  -o yaml > myapp.yaml
$ vi myapp.yaml
$ oc create -f myapp.yaml
```

2.3.4.5. 別名でのオブジェクトの作成

通常 **new-app** で作成されるオブジェクトの名前はソースリポジトリまたは生成に使用されたイメージに基づいて付けられます。コマンドに **--name** フラグを追加することで、生成されたオブジェクトの名前を設定できます。

```
$ oc new-app https://github.com/openshift/ruby-hello-world --name=myapp
```

2.3.4.6. 別のプロジェクトでのオブジェクトの作成

通常 **new-app** は現在のプロジェクトにオブジェクトを作成します。ただし、**-n|--namespace** 引数を使用して、別のプロジェクトにオブジェクトを作成することができます。

```
$ oc new-app https://github.com/openshift/ruby-hello-world -n myproject
```

2.3.4.7. 複数のオブジェクトの作成

new-app コマンドは、複数のパラメーターを **new-app** に指定して複数のアプリケーションを作成できます。コマンドラインで指定するラベルは、単一コマンドで作成されるすべてのオブジェクトに適用されます。環境変数は、ソースまたはイメージから作成されたすべてのコンポーネントに適用されます。

ソースリポジトリおよび Docker Hub イメージからアプリケーションを作成するには、以下を実行します。

```
$ oc new-app https://github.com/openshift/ruby-hello-world mysql
```



注記

ソースコードリポジトリおよびビルダーイメージが別個の引数として指定されている場合、**new-app** はソースコードリポジトリのビルダーとしてそのビルダーイメージを使用します。これを意図していない場合は、**~**セパレーターを使用してソースに必要なビルダーイメージを指定します。

2.3.4.8. 単一 Pod でのイメージとソースのグループ化

new-app コマンドにより、単一 Pod に複数のイメージをまとめてデプロイできます。イメージのグループ化を指定するには **+**セパレーターを使用します。**--group** コマンドライン引数をグループ化する必要のあるイメージを指定する際にも使用することもできます。ソースリポジトリからビルドされたイメージを別のイメージと共にグループ化するには、そのビルダーイメージをグループで指定します。

```
$ oc new-app ruby+mysql
```

ソースからビルドされたイメージと外部のイメージをまとめてデプロイするには、以下を実行します。

```
$ oc new-app \
  ruby~https://github.com/openshift/ruby-hello-world \
  mysql \
  --group=ruby+mysql
```

2.3.4.9. イメージ、テンプレート、および他の入力の検索

イメージ、テンプレート、および **oc new-app** コマンドの他の入力内容を検索するには、**--search** フラグおよび **--list** フラグを追加します。たとえば、PHP を含むすべてのイメージまたはテンプレートを検索するには、以下を実行します。

```
$ oc new-app --search php
```

2.4. TOPOLOGY ビューを使用したアプリケーション構成の表示

Web コンソールの **Developer** パースペクティブにある **Topology** ビューは、プロジェクト内のすべてのアプリケーション、それらのビルドステータスおよびアプリケーションに関連するコンポーネントとサービスを視覚的に表示します。


前提条件

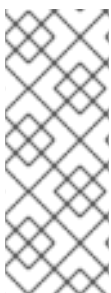
Topology ビューでアプリケーションを表示し、それらと対話するには、以下を確認します。

- [Web コンソールにログイン](#)している。
- **Developer** パースペクティブにいる。
- OpenShift Container Platform でアプリケーションおよび他のワークロードを作成するための適切なプロジェクト内の[ロールおよびパーミッション](#)がある。
- **Developer** パースペクティブを使用して [OpenShift Container Platform](#) でアプリケーションを作成し、デプロイしている。

2.4.1. アプリケーションのトポロジーの表示







Developer パースペクティブの左側のナビゲーションパネルを使用すると、**Topology** ビューに移動できます。アプリケーションを作成したら、**Topology** ビューに自動的に移動します。ここでは、アプリケーション Pod のステータスの確認、パブリック URL でのアプリケーションへの迅速なアクセス、ソースコードへのアクセスとその変更、最終ビルドのステータスの確認ができます。ズームインおよびズームアウトにより、特定のアプリケーションの詳細を表示することができます。

サーバーレスアプリケーションは、Knative シンボルで視覚的に表示されます ()。

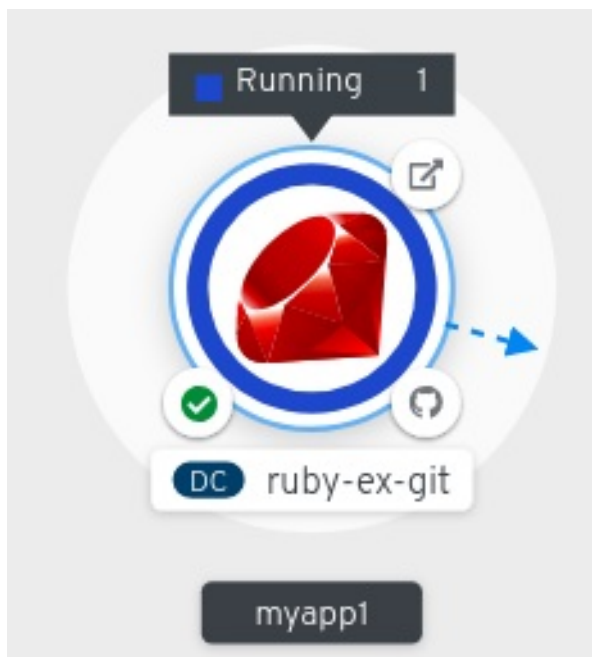


注記

サーバーレスアプリケーションは、**Topology** ビューでの読み込みおよび表示にしばらく時間がかかります。サーバーレスアプリケーションを作成すると、これは最初にサービスリソースを作成し、次にリビジョンを作成します。続いてデプロイされて **Topology** ビューに表示されます。これが唯一のワークロードの場合には、**Add** ページにリダイレクトされる可能性があります。リビジョンがデプロイされると、サーバーレスアプリケーションは **Topology** ビューに表示されます。

Pod のステータスまたはフェーズは、色で区別され、ツールチップで次のように表示されます。Running (), Not Ready (), Warning (), Failed (), Pending (), Succeeded (), Terminating (), または Unknown ()。Pod のステータスについての詳細は、[Kubernetes ドキュメント](#) を参照してください。

アプリケーションを作成し、イメージがデプロイされると、ステータスは **Pending** と表示されます。アプリケーションをビルドすると、**Running** と表示されます。



以下のように、異なるタイプのリソースオブジェクトのインジケータと共に、アプリケーションリソース名が追加されます。

- **DC:** DeploymentConfigs
- **D:** Deployment
- **SS:** StatefulSet
- **DS:** Daemonset

2.4.2. アプリケーションとの対話

Web コンソールの **Developer** パースペクティブの **Topology** ビューは、アプリケーションと対話するための以下のオプションを提供します。

- **Open URL** (🔗) をクリックして、パブリック URL のルートで公開されるアプリケーションを表示します。
- **Edit Source code** をクリックして、ソースコードにアクセスし、これを変更します。



注記

この機能は、**From Git**、**From Catalog**、および **From Dockerfile** オプションを使用してアプリケーションを作成する場合にのみ利用できます。

Eclipse Che Operator がクラスターにインストールされている場合、Che ワークスペースが作成され、ソースコードを編集するためにワークスペースにダイレクトされます。インストールされていない場合には、ソースコードがホストされている Git リポジトリ (🔄) にダイレクトされます。

- カーソルを Pod の左下のアイコンの上に置き、最新ビルドおよびそのステータスを確認します。アプリケーションビルドのステータスは次のように表示されます。 **New** (🕒)、**Pending** (🕒)、**Running** (🔄)、**Completed** (✅)、**Failed** (❌)、および **Canceled** (🚫)。

2.4.3. アプリケーション Pod のスケーリングおよびビルドとルートの確認

Topology ビューは、**Overview** パネルでデプロイ済みのコンポーネントの詳細を提供します。**Overview** および **Resources** タブを使用して、アプリケーション Pod をスケーリングし、ビルドのステータス、サービスおよびルートについて以下のように確認できます。

- コンポーネントノードをクリックし、右側の **Overview** パネルを確認します。**Overview** タブを使用して、以下を実行します。
 - 上下の矢印を使用して Pod をスケーリングし、アプリケーションのインスタンス数の増減を手動で調整します。サーバーレスアプリケーションの場合、Pod は、チャンネルのトラフィックに基づいてアイドルおよびスケールアップ時に自動的にゼロにスケーリングされます。
 - アプリケーションの **ラベル**、**アノテーション** および **ステータス** を確認します。
- **Resources** タブをクリックして、以下を実行します。
 - すべての Pod の一覧を確認し、それらのステータスを表示し、ログにアクセスし、Pod をクリックして Pod の詳細を表示します。
 - ビルド、ステータスを確認し、ログにアクセスし、必要に応じて新規ビルドを開始します。
 - コンポーネントによって使用されるサービスとルートを確認します。

サーバーレスアプリケーションの場合、**Resources** タブは、そのコンポーネントに使用されるリビジョン、ルート、および設定に関する情報を提供します。

2.4.4. アプリケーション内での複数コンポーネントのグループ化

Add ページを使用して、複数のコンポーネントまたはサービスをプロジェクトに追加し、**Topology** ページを使用してアプリケーショングループ内のアプリケーションとリソースをグループ化できます。以下の手順では、MongoDB データベースサービスを Node.js コンポーネントを使用して既存のアプリケーションに追加します。

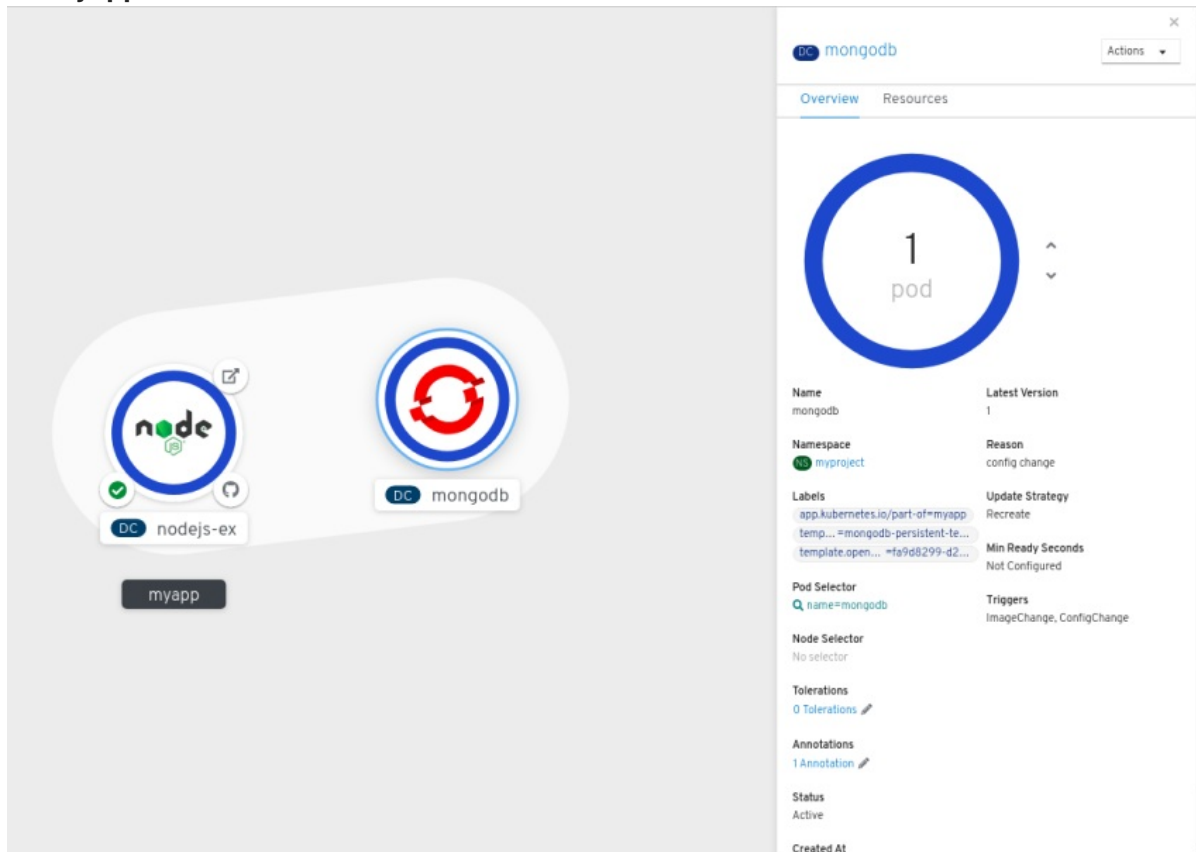
前提条件

- **Developer** パースペクティブを使用して、OpenShift Container Platform に Node.js アプリケーションを作成し、デプロイしている。

手順

1. 以下のように MongoDB サービスを作成し、これをプロジェクトにデプロイします。
 - a. **Developer** パースペクティブで、**Add** ビューに移動して **Database** オプションを選択し、**Developer Catalog**を確認します。ここには、アプリケーションにコンポーネントまたはサービスとして追加できる複数のオプションがあります。
 - b. **MongoDB** オプションをクリックし、サービスの詳細を確認します。
 - c. **Instantiate Template** をクリックして、MongoDB サービスの詳細情報を含む自動的に設定されたテンプレートを表示し、**Create** をクリックしてサービスを作成します。
2. 左側のナビゲーションパネルで **Topology** をクリックし、プロジェクトにデプロイされた MongoDB サービスを表示します。

3. MongoDB サービスを既存のアプリケーショングループに追加するには、**mongodb** Pod を選択して、これをアプリケーションにドラッグします。MongoDB サービスは既存のアプリケーショングループに追加されます。
4. コンポーネントをドラッグし、これをアプリケーショングループに追加すると、必要なラベルがコンポーネントに自動的に追加されます。MongoDB サービスノードをクリックし、**Overview** パネルの **Labels** セクションに追加されたラベル **app.kubernetes.io/part-of=myapp** を確認します。



または、以下のようにコンポーネントをアプリケーションに追加することもできます。

1. MongoDB サービスをアプリケーションに追加するには、**mongodb** Pod をクリックし、右側の **Overview** パネルを確認します。
2. パネルの右上にある **Actions** ドロップダウンメニューをクリックし、**Edit Application Grouping** を選択します。
3. **Edit Application Grouping** ダイアログボックスで、**Select an Application** ドロップダウンリストをクリックし、適切なアプリケーショングループを選択します。
4. **Save** をクリックし、アプリケーショングループに追加された MongoDB サービスを表示します。

2.4.5. アプリケーション内および複数のアプリケーション間でのコンポーネントの接続

アプリケーション内で複数のコンポーネントをグループ化することに加え、**Topology** ビューを使用してコンポーネントを相互に接続することもできます。MongoDB サービスは、以下のように Node.js アプリケーションを使用して接続できます。

前提条件

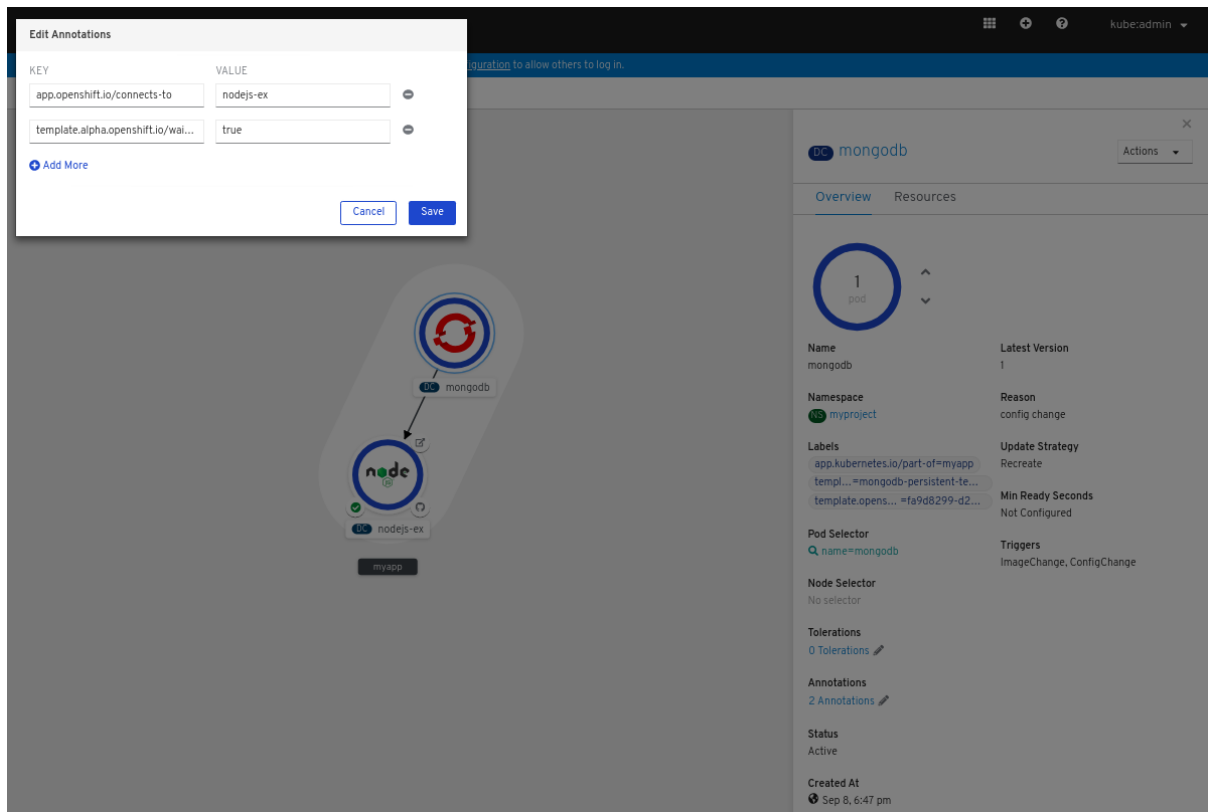
- **Developer** パースペクティブを使用して、OpenShift Container Platform に Node.js アプリケーションを作成し、デプロイしている。
- **Developer** パースペクティブを使用して、OpenShift Container Platform に MongoDB サービスを作成し、デプロイしている。

手順

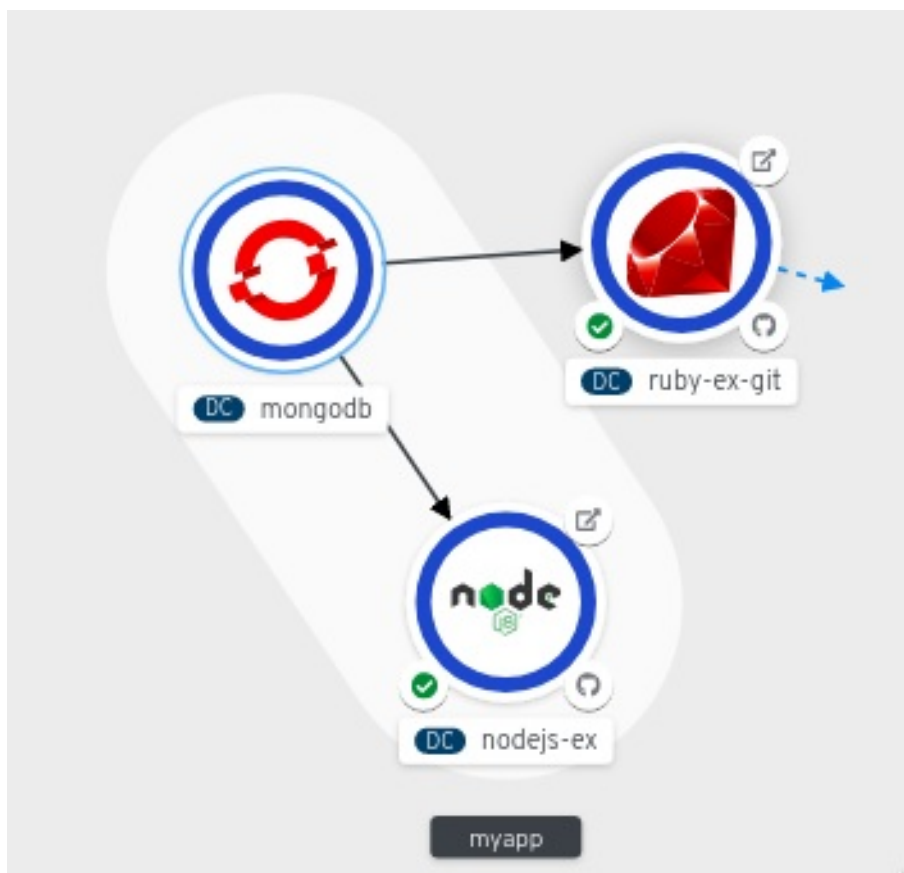
1. カーソルを MongoDB サービスの上に置き、ノードの矢印を確認します。



2. 矢印をクリックして Node.js コンポーネントの方にドラッグし、MongoDB サービスをこれに接続します。
3. MongoDB サービスをクリックし、**Overview** パネルを表示します。**Annotations** セクションで、編集アイコンをクリックして **Key = app.openshift.io/connects-to** および **Value = nodejs-ex** アノテーションがサービスに追加されていることを確認します。



他のアプリケーションやコンポーネントを同様に作成し、それらの間の接続を確立することができます。



2.4.6. Topology ビューに使用するラベルとアノテーション

Topology ビューは、以下のラベルおよびアノテーションを使用します。

ノードに表示されるアイコン

ノードのアイコンは、最初に **app.openshift.io/runtime** ラベルを使用してから **app.kubernetes.io/name** ラベルを使用して一致するアイコンを検索して定義されます。このマッチングは、事前定義されたアイコンセットを使用して行われます。

ソースコードエディターまたはソースへのリンク

app.openshift.io/vcs-uri アノテーションは、ソースコードエディターへのリンクを作成するために使用されます。

ノードコネクタ

app.openshift.io/connects-to アノテーションは、ノードに接続するために使用されます。

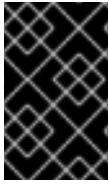
アプリケーションのグループ化

app.kubernetes.io/part-of=<appname> ラベルは、アプリケーション、サービス、およびコンポーネントをグループ化するために使用されます。

OpenShift Container Platform アプリケーションで使用する必要のあるラベルとアノテーションの詳細については、「[Guidelines for labels and annotations for OpenShift applications](#)」を参照してください。

第3章 サービスブローカー

3.1. サービスカタログのインストール



重要

サービスカタログは OpenShift Container Platform 4 では非推奨になっています。同等または強化された機能は Operator Framework および Operator Lifecycle Manager (OLM) で提供されます。

3.1.1. サービスカタログについて

マイクロサービスベースのアプリケーションを開発して、クラウドネイティブのプラットフォームで実行する場合には、サービスプロバイダーやプラットフォームに合わせて、異なるリソースをプロビジョニングし、その位置情報 (coordinate)、認証情報および設定を共有する数多くの方法を利用できます。

開発者がよりシームレスに作業できるように、OpenShift Container Platform には Kubernetes 向けの [Open Service Broker API \(OSB API\)](#) の実装である **サービスカタログ** が含まれています。これにより、OpenShift Container Platform にデプロイされているアプリケーションをさまざまな種類のサービスブローカーに接続できます。

サービスカタログでは、クラスター管理者が1つの API 仕様を使用して、複数のプラットフォームを統合できます。OpenShift Container Platform Web コンソールは、サービスカタログにサービスブローカーによって提供されるクラスターサービスカタログを表示するので、ユーザーはこれらのサービスをそれぞれのアプリケーションで使用できるようにサービスの検出やインスタンス化を実行できます。

結果として、サービスのユーザーは異なるプロバイダーが提供する異なるタイプのサービスを簡単かつ一貫して使用できるという利点が得られます。また、サービスプロバイダーは、複数のプラットフォームにアクセスできる統合ポイントという利点を得られます。

サービスカタログは、OpenShift Container Platform 4 ではデフォルトでインストールされません。

3.1.2. サービスカタログのインストール

OpenShift Ansible Broker またはテンプレートサービスブローカーからサービスを使用することを計画する場合、以下の手順を実行してサービスカタログをインストールする必要があります。

サービスカタログの API サーバーおよびコントローラーマネージャーのカスタムリソースは OpenShift Container Platform にデフォルトで作成されますが、初期の状態は **managementState** が **Removed** になります。サービスカタログをインストールするには、それらのリソースの **managementState** を **Managed** に変更する必要があります。

手順

1. サービスカタログ API サーバーを有効にします。
 - a. 以下のコマンドを使用して、サービスカタログ API サーバーリソースを編集します。

```
$ oc edit servicecatalogapiservers
```

- b. **spec** の下で、**managementState** フィールドを **Managed** に設定します。

```
spec:
  logLevel: Normal
  managementState: Managed
```

- c. 変更を適用するためにファイルを保存します。
Operator はサービスカタログ API サーバーコンポーネントをインストールします。
OpenShift Container Platform 4 の時点では、このコンポーネントは **openshift-service-catalog-apiser** namespace にインストールされます。
2. サービスカタログコントローラマネージャーを有効にします。
 - a. 以下のコマンドを使用して、サービスカタログコントローラマネージャーのリソースを編集します。

```
$ oc edit servicecatalogcontrollermanagers
```

- b. **spec** の下で、**managementState** フィールドを **Managed** に設定します。

```
spec:
  logLevel: Normal
  managementState: Managed
```

- c. 変更を適用するためにファイルを保存します。
Operator はサービスカタログコントローラマネージャーをインストールします。
OpenShift Container Platform 4 の時点で、このコンポーネントは **openshift-service-catalog-controller-manager** namespace にインストールされます。

3.2. テンプレートサービスブローカーのインストール

テンプレートサービスブローカーをインストールし、これが提供するテンプレートアプリケーションへのアクセスを取得します。



重要

テンプレートサービスブローカーは OpenShift Container Platform 4 では非推奨になっています。同等または強化された機能は Operator Framework および Operator Lifecycle Manager (OLM) で提供されます。

前提条件

- [サービスカタログのインストール](#)

3.2.1. テンプレートサービスブローカーについて

テンプレートサービスブローカーは、サービスカタログに対し、初期リリース以降 OpenShift Container Platform に同梱されるデフォルトのインスタントアプリケーションおよびクイックスタートテンプレートを可視化します。さらにテンプレートサービスブローカーは、Red Hat、クラスター管理者、またはユーザーないしはサードパーティーベンダーのいずれかが作成する OpenShift Container Platform テンプレートのいずれのコンテンツもサービスとして利用可能にすることができます。

デフォルトで、テンプレートサービスブローカーは **openshift** プロジェクトからグローバルに利用できるオブジェクトを表示します。また、これはクラスター管理者が選択する他のプロジェクトを監視するように設定することもできます。

テンプレートサービスブローカーは、OpenShift Container Platform 4 ではデフォルトでインストールされません。

3.2.2. テンプレートサービスブローカー Operator のインストール

前提条件

- サービスカタログがインストールされていること。

手順

以下の手順では、Web コンソールを使用してテンプレートサービスブローカー Operator をインストールします。

1. namespace を作成します。
 - a. Web コンソールで **Administration** → **Namespaces** に移動し、**Create Namespace** をクリックします。
 - b. **Name** フィールドに **openshift-template-service-broker** を入力し、**Create** をクリックします。



注記

namespace は **openshift-** で開始する必要があります。

2. **Operators** → **OperatorHub** ページに移動します。 **openshift-template-service-broker** プロジェクトが選択されていることを確認します。
3. **Template Service Broker Operator** を選択します。
4. Operator についての情報を確認してから、**Install** をクリックします。
5. デフォルトの選択を確認し、**Subscribe** をクリックします。

次に、テンプレートサービスブローカーを起動し、これが提供するテンプレートアプリケーションへのアクセスを取得します。

3.2.3. テンプレートサービスブローカーの起動

テンプレートサービスブローカー Operator のインストール後に、以下の手順でテンプレートサービスブローカーを起動します。

前提条件

- サービスカタログがインストールされていること。
- テンプレートサービスブローカー Operator がインストールされていること。

手順

1. Web コンソールで **Operators** → **Installed Operators** に移動し、 **openshift-template-service-broker** プロジェクトを選択します。
2. **Template Service Broker Operator** を選択します。

3. **Provided APIs** で、**Template Service Broker**について **Create New** をクリックします。
4. デフォルトのYAMLを確認し、**Create** をクリックします。
5. テンプレートサービスブローカーが起動していることを確認します。
 テンプレートサービスブローカーの起動後に、**Catalog** → **Developer Catalog** に移動し、**Service Class** チェックボックスを選択して利用可能なテンプレートアプリケーションを表示できます。テンプレートサービスブローカーが起動し、テンプレートアプリケーションが利用可能になるまで数分の時間がかかる場合があります。

これらのサービスクラスが表示されない場合は、以下の項目のステータスを確認できます。

- テンプレートサービスブローカー Pod のステータス
 - **openshift-template-service-broker** プロジェクトの **Workloads** → **Pods** ページから、**apiserver-** で起動する Pod のステータスが **Running** であり、**Ready** の準備状態であることを確認します。
- クラスタサービスブローカーのステータス
 - **Catalog** → **Broker Management** → **Service Brokers** ページから、**template-service-broker** サービスブローカーのステータスが **Ready** であることを確認します。
- サービスカタログコントローラマネージャー Pod のログ
 - **openshift-service-catalog-controller-manager** プロジェクトの **Workloads** → **Pods** ページから、それぞれの Pod のログを確認し、**Successfully fetched catalog entries from broker** のメッセージと共にログエントリが表示されていることを確認します。

3.3. テンプレートアプリケーションのプロビジョニング

3.3.1. テンプレートアプリケーションのプロビジョニング

以下の手順では、テンプレートサービスブローカーによって利用可能にされたサンプル PostgreSQL テンプレートアプリケーションをプロビジョニングします。

前提条件

- サービスカタログがインストールされていること。
- テンプレートサービスブローカーがインストールされていること。

手順

1. プロジェクトを作成します。
 - a. Web コンソールで、**Home** → **Projects** に移動し、**Create Project** をクリックします。
 - b. **test-postgresql** を **Name** フィールドに入力し、**Create** をクリックします。
2. サービスインスタンスを作成します。
 - a. **Catalog** → **Developer Catalog** ページに移動します。
 - b. **PostgreSQL (Ephemeral)** テンプレートアプリケーションを選択し、**Create Service Instance** をクリックします。

- c. デフォルトの選択を確認し、それ以外の必要なフィールドを設定してから **Create** をクリックします。
 - d. **Catalog** → **Provisioned Services** に移動し、**postgresql-ephemeral** サービスインスタンスが作成され、ステータスが **Ready** であることを確認します。
Home → **Events** ページで進捗を確認できます。しばらくすると、**postgresql-ephemeral** のイベントが「The instance was provisioned successfully」というメッセージと共に表示されるはずです。
3. サービスバインディングを作成します。
 - a. **Provisioned Services** ページから、**postgresql-ephemeral** をクリックし、**Create Service Binding** をクリックします。
 - b. デフォルトのサービスバインディング名を確認し、**Create** をクリックします。
これにより、指定された名前を使用してバインディングの新規シークレットが作成されます。
 4. 作成されたシークレットを確認します。
 - a. **Workloads** → **Secrets** に移動し、**postgresql-ephemeral** という名前のシークレットが作成されていることを確認します。
 - b. **postgresql-ephemeral** をクリックし、他のアプリへのバインディングに使用されるキーと値のペアを **Data** セクションで確認します。

3.4. テンプレートサービスブローカーのアンインストール

テンプレートサービスブローカーは、これが提供するテンプレートアプリケーションへのアクセスを必要としなくなった場合にアンインストールできます。



重要

テンプレートサービスブローカーは OpenShift Container Platform 4 では非推奨になっています。同等または強化された機能は Operator Framework および Operator Lifecycle Manager (OLM) で提供されます。

3.4.1. テンプレートサービスブローカーのアンインストール

以下の手順では、Web コンソールを使用してテンプレートサービスブローカーおよび Operator をアンインストールします。



警告

クラスターにテンプレートサービスブローカーからプロビジョニングされたサービスがある場合には、これをアンインストールしないでください。アンインストールすると、サービスを管理しようとする際にエラーが生じる可能性があります。

前提条件

- テンプレートサービスブローカーがインストールされていること。

手順

この手順では、テンプレートサービスブローカーが **openshift-template-service-broker** プロジェクトにインストールされていることを前提とします。

1. テンプレートサービスブローカーのアンインストール
 - a. **Operators** → **Installed Operators** に移動し、ドロップダウンメニューから **openshift-template-service-broker** プロジェクトを選択します。
 - b. **Template Service Broker Operator** をクリックします。
 - c. **Template Service Broker** タブを選択します。
 - d. **template-service-broker** をクリックします。
 - e. **Actions** ドロップダウンメニューから、**Delete Template Service Broker** を選択します。
 - f. 確認ポップアップ画面から **Delete** をクリックします。
テンプレートサービスブローカーのアンインストールが終了し、テンプレートアプリケーションは Developer Catalog からすぐに削除されます。
2. テンプレートサービスブローカー Operator のアンインストール
 - a. **Operators** → **Installed Operators** ページに移動し、**Filter by name** にスクロールするか、またはキーワードを入力して **Template Service Broker Operator** 検索し、これをクリックします。
 - b. **Operator Details** ページの右側で、**Actions** ドロップダウンメニューから **Uninstall Operator** を選択します。
 - c. **Remove Operator Subscription** ウィンドウでプロンプトが表示されたら、インストールに関連するすべてのコンポーネントを削除する場合は、**Also completely remove the Operator from the selected namespace** チェックボックスをオプションで選択します。これにより CSV が削除され、次に Operator に関連付けられた Pod、Deployment、CRD および CR が削除されます。
 - d. **Remove** を選択します。この Operator は実行を停止し、更新を受信しなくなります。テンプレートサービスブローカー Operator がクラスターにインストールされていない状態になります。

テンプレートサービスブローカーのアンインストール後に、ユーザーはテンプレートサービスブローカーによって提供されるテンプレートアプリケーションにアクセスできなくなります。

3.5. OPENSIFT ANSIBLE BROKER のインストール

OpenShift Ansible Broker をインストールし、これが提供するサービスバンドルへのアクセスを取得します。



重要

OpenShift Ansible Broker は OpenShift Container Platform 4 では非推奨になっていません。同等または強化された機能は Operator Framework および Operator Lifecycle Manager (OLM) で提供されます。

前提条件

- [サービスカタログのインストール](#)

3.5.1. OpenShift Ansible Broker について

OpenShift Ansible Broker は、Ansible playbook bundles (APB) で定義されるアプリケーションを管理する Open Service Broker (OSB) API の実装です。APB は、OpenShift Container Platform のコンテナアプリケーションを定義し、配信する方法を提供し、Ansible ランタイムと共にコンテナイメージに組み込まれた Ansible Playbook のバンドルで構成されています。APB は Ansible を活用し、複雑なデプロイメントを自動化する標準メカニズムを構築します。

OpenShift Ansible Brokerは、以下の基本的なワークフローに従います。

1. ユーザーは、OpenShift Container Platform Web コンソールを使用してサービスカタログから利用可能なアプリケーションの一覧を要求します。
2. サービスカタログは、OpenShift Ansible Broker から利用可能なアプリケーションの一覧を要求します。
3. OpenShift Ansible Broker は定義されたコンテナイメージレジストリーと通信し、利用可能な APB の情報を得ます。
4. ユーザーは特定の APB をプロビジョニングする要求を実行します。
5. OpenShift Ansible Broker は、APB でプロビジョニングメソッドを呼び出して、ユーザーのプロビジョニング要求に対応します。

OpenShift Ansible Broker は、OpenShift Container Platform 4 ではデフォルトでインストールされません。

3.5.1.1. Ansible Playbook Bundle

Ansible Playbook Bundle (APB) は、Ansible ロールおよび Playbook の既存の投資を活用できるようにする軽量アプリケーション定義です。

APB は、名前の付いた Playbook が含まれる単純なディレクトリーを使用し、プロビジョニングやバインドなどの OSB API アクションを実行します。**apb.yml** ファイルで定義するメタデータには、デプロイメント時に使用する必須/任意のパラメーターの一覧が含まれています。

追加リソース

- [Ansible Playbook Bundle リポジトリー](#)

3.5.2. OpenShift Ansible Service Broker Operator のインストール

前提条件

- サービスカタログがインストールされていること。

手順

以下の手順では、Web コンソールを使用して OpenShift Ansible Service Broker Operator をインストールします。

1. namespace を作成します。

- a. Web コンソールで **Administration** → **Namespaces** に移動し、**Create Namespace** をクリックします。
- b. **openshift-ansible-service-broker** を **Name** フィールドに、**openshift.io/cluster-monitoring=true** を **Labels** フィールドに入力し、**Create** をクリックします。



注記

namespace は **openshift-** で開始する必要があります。

2. クラスターのロールバインディングを作成します。
 - a. **Administration** → **Role Bindings** に移動し、**Create Binding** をクリックします。
 - b. **Binding Type** については、**Cluster-wide Role Binding (ClusterRoleBinding)** を選択します。
 - c. **Role Binding** については、**ansible-service-broker** を **Name** フィールドに入力します。
 - d. **Role** については、**admin** を選択します。
 - e. **Subject** については、**Service Account** オプションを選択し、**openshift-ansible-service-broker** namespace を選択して **openshift-ansible-service-broker-operator** を **Subject Name** フィールドに入力します。
 - f. **Create** をクリックします。
3. Red Hat Container Catalog に接続するためにシークレットを作成します。
 - a. **Workloads** → **Secrets** に移動します。**openshift-ansible-service-broker** プロジェクトが選択されていることを確認します。
 - b. **Create** → **Key/Value Secret** をクリックします。
 - c. **asb-registry-auth** を **シークレット名** として入力します。
 - d. **username** の **Key** および Red Hat Container Catalog ユーザー名の **Value** を追加します。
 - e. **Add Key/Value** をクリックし、**password** の **Key** および Red Hat Container Catalog パスワードの **Value** を追加します。
 - f. **Create** をクリックします。
4. **Operators** → **OperatorHub** ページに移動します。**openshift-ansible-service-broker** プロジェクトが選択されていることを確認します。
5. **OpenShift Ansible Service Broker Operator** を選択します。
6. Operator についての情報を確認してから、**Install** をクリックします。
7. デフォルトの選択を確認し、**Subscribe** をクリックします。

次に、OpenShift Ansible Broker を起動し、これが提供するサービスバンドルへのアクセスを取得します。

3.5.3. OpenShift Ansible Broker の起動

OpenShift Ansible Broker Operator のインストール後に、以下の手順で OpenShift Ansible Broker を起動します。

前提条件

- サービスカタログがインストールされていること。
- OpenShift Ansible Service Broker Operator がインストールされていること。

手順

1. Web コンソールで **Operators** → **Installed Operators** に移動し、**openshift-ansible-service-broker** プロジェクトを選択します。
2. **OpenShift Ansible Service Broker Operator** を選択します。
3. **Provided APIs** で、**Automation Broker** について **Create New** をクリックします。
4. 以下を、提供されているデフォルトの YAML の **spec** フィールドに追加します。

```
registry:
  - name: rhcc
    type: rhcc
    url: https://registry.redhat.io
    auth_type: secret
    auth_name: asb-registry-auth
```

これは、OpenShift Ansible Service Broker Operator のインストール時に作成されたシークレットを参照します。これにより、Red Hat Container Catalog に接続できます。

5. 追加の OpenShift Ansible Broker 設定オプションを設定し、**Create** をクリックします。
6. OpenShift Ansible Broker が起動したことを確認します。
OpenShift Ansible Broker の起動後に、**Catalog** → **Developer Catalog** に移動し、**Service Class** チェックボックスを選択して利用可能なサービスバンドルを表示できます。OpenShift Ansible Broker が起動し、サービスバンドルが利用可能になるまで数分の時間がかかる場合があります。

これらのサービスクラスが表示されない場合は、以下の項目のステータスを確認できます。

- OpenShift Ansible Broker Pod のステータス
 - **openshift-ansible-service-broker** プロジェクトの **Workloads** → **Pods** ページから、**asb-** で起動する Pod のステータスが **Running** であり、**Ready** の準備状態であることを確認します。
- クラスターサービスブローカーのステータス
 - **Catalog** → **Broker Management** → **Service Brokers** ページから、**ansible-service-broker** サービスブローカーのステータスが **Ready** であることを確認します。
- サービスカタログコントローラーマネージャー Pod のログ
 - **openshift-service-catalog-controller-manager** プロジェクトの **Workloads** → **Pods** ページから、それぞれの Pod のログを確認し、**Successfully fetched catalog entries from broker** のメッセージと共にログエントリが表示されていることを確認します。

3.5.3.1. OpenShift Ansible Broker 設定オプション

OpenShift Ansible Broker の以下のオプションを設定できます。

表3.1 OpenShift Ansible Broker 設定オプション

YAML キー	説明	デフォルト値
brokerName	Broker インスタンスを特定するために使用される名前。	ansible-service-broker
brokerNamespace	Broker が置かれている namespace。	openshift-ansible-service-broker
brokerImage	Broker に使用されている完全修飾イメージ。	docker.io/ansibleplaybookbundle/origin-ansible-service-broker:v4.0
brokerImagePullPolicy	Broker イメージ自体に使用されるプルポリシー。	IfNotPresent
brokerNodeSelector	Broker のデプロイメントに使用されるノードセレクター文字列。	"
registries	Broker レジストリー設定の yaml 一覧として表現される。これにより、ユーザーは Broker が検出し、APB の取得に使用するイメージレジストリーを設定できます。	デフォルトレジストリー配列 を参照してください。
logLevel	Broker のログに使用されるログレベル。	info
apbPullPolicy	APB Pod に使用されるプルポリシー。	IfNotPresent
sandboxRole	APB を実行するために使用されるサービスアカウントに付与されるロール。	edit
keepNamespace	APB の完了後に APB を実行するために作成された一時 namespace が削除されたかどうか (結果の如何は問わない)。	false
keepNamespaceOnError	結果がエラーの場合のみ、APB の完了後に APB を実行するために作成された一時 namespace が削除されたかどうか。	false
bootstrapOnStartup	Broker が起動時にブートストラップルーチンを実行する必要があるかどうか。	true
refreshInterval	APB のインベントリーを更新する Broker ブートストラップ間隔。	600s

YAML キー	説明	デフォルト値
launchApbOnBind	Experimental: バインド操作時に APB を実行する Broker を切り替えます。	false
autoEscalate	APB の実行中に、Broker がユーザーのパーミッションをエスカレートするかどうか。これは、Broker が起点となるユーザー承認を実行してユーザーが APB サンドボックスに付与されたパーミッションを持つようにするため、通常は false のままになります。	false
outputRequest	Broker が受信する低レベル HTTP 要求を出力するかどうか。	false

registries のデフォルト配列

```
- type: rhcc
  name: rhcc
  url: https://registry.redhat.io
  white_list:
  - ".*-apb$"
  auth_type: secret
  auth_name: asb-registry-auth
```

3.6. OPENSIFT ANSIBLE BROKER の設定



重要

OpenShift Ansible Broker は OpenShift Container Platform 4 では非推奨になっています。同等または強化された機能は Operator Framework および Operator Lifecycle Manager (OLM) で提供されます。

3.6.1. OpenShift Ansible Broker の設定

以下の手順では、OpenShift Ansible Broker の設定をカスタマイズします。

前提条件

- OpenShift Ansible Broker がインストールされていること。

手順

この手順では、**ansible-service-broker** を OpenShift Ansible Broker 名前とインストール先のプロジェクトの両方に使用していることを前提とします。

1. Web コンソールで **Operators** → **Installed Operators** に移動し、**ansible-service-broker** プロジェクトを選択します。
2. **OpenShift Ansible Service Broker Operator** を選択します。
3. **Automation Broker** タブで、**ansible-service-broker** を選択します。

4. YAML タブの **spec** フィールドの下で OpenShift Ansible Broker 設定オプションを追加するか、または更新します。
以下は例になります。

```
spec:
  keepNamespace: true
  sandboxRole: edit
```

5. Save をクリックして変更を適用します。

3.6.1.1. OpenShift Ansible Broker 設定オプション

OpenShift Ansible Broker の以下のオプションを設定できます。

表3.2 OpenShift Ansible Broker 設定オプション

YAML キー	説明	デフォルト値
brokerName	Broker インスタンスを特定するために使用される名前。	ansible-service-broker
brokerNamespace	Broker が置かれている namespace。	openshift-ansible-service-broker
brokerImage	Broker に使用されている完全修飾イメージ。	docker.io/ansibleplaybookbundle/origin-ansible-service-broker:v4.0
brokerImagePullPolicy	Broker イメージ自体に使用されるプルポリシー。	IfNotPresent
brokerNodeSelector	Broker のデプロイメントに使用されるノードセレクター文字列。	"
registries	Broker レジストリー設定の yaml 一覧として表現される。これにより、ユーザーは Broker が検出し、APB の取得に使用するイメージレジストリーを設定できます。	デフォルトレジストリー配列 を参照してください。
logLevel	Broker のログに使用されるログレベル。	info
apbPullPolicy	APB Pod に使用されるプルポリシー。	IfNotPresent
sandboxRole	APB を実行するために使用されるサービスアカウントに付与されるロール。	edit
keepNamespace	APB の完了後に APB を実行するために作成された一時 namespace が削除されたかどうか (結果の如何は問わない)。	false

YAML キー	説明	デフォルト値
keepNamespaceOnError	結果がエラーの場合のみ、APB の完了後に APB を実行するために作成された一時 namespace が削除されたかどうか。	false
bootstrapOnStartup	Broker が起動時にブートストラップルーチンを実行する必要があるかどうか。	true
refreshInterval	APB のインベントリを更新する Broker ブートストラップ間隔の間隔。	600s
launchApbOnBind	Experimental: バインド操作時に APB を実行する Broker を切り替えます。	false
autoEscalate	APB の実行中に、Broker がユーザーのパーミッションをエスカレートするかどうか。これは、Broker が起点となるユーザー承認を実行してユーザーが APB サンドボックスに付与されたパーミッションを持つようにするため、通常は false のままになります。	false
outputRequest	Broker が受信する低レベル HTTP 要求を出力するかどうか。	false

registries のデフォルト配列

```
- type: rhcc
  name: rhcc
  url: https://registry.redhat.io
  white_list:
  - ".*-apb$"
  auth_type: secret
  auth_name: asb-registry-auth
```

3.6.2. OpenShift Ansible Broker のモニタリング設定

Prometheus が OpenShift Ansible Broker をモニターできるようにするには、以下のリソースを作成して、OpenShift Ansible Broker がインストールされている namespace にアクセスできるように Prometheus にパーミッションを付与する必要があります。

前提条件

- OpenShift Ansible Broker がインストールされていること。



注記

この手順では、OpenShift Ansible Broker が **openshift-ansible-service-broker** namespace にインストールされていることを前提とします。

手順

1. ロールを作成します。
 - a. **Administration** → **Roles** に移動し、**Create Role** をクリックします。
 - b. エディターで YAML を以下に置き換えます。

```

apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: prometheus-k8s
  namespace: openshift-ansible-service-broker
rules:
- apiGroups:
  - ""
  resources:
  - services
  - endpoints
  - pods
  verbs:
  - get
  - list
  - watch

```

- c. **Create** をクリックします。
2. ロールバインディングを作成します。
 - a. **Administration** → **Role Bindings** に移動し、**Create Binding** をクリックします。
 - b. **Binding Type** について、**Namespace Role Binding (RoleBinding)** を選択します。
 - c. **Role Binding** について、**prometheus-k8s** を **Name** フィールドに、**openshift-ansible-service-broker** を **Namespace** フィールドに入力します。
 - d. **Role** について、**prometheus-k8s** を選択します。
 - e. **Subject** について、**Service Account** オプションを選択し、**openshift-monitoring namespace** を選択してから **prometheus-k8s** を **Subject Name** フィールドに入力します。
 - f. **Create** をクリックします。

Prometheus は OpenShift Ansible Broker メトリクスにアクセスできるようになります。

3.7. サービスバンドルのプロビジョニング

3.7.1. サービスバンドルのプロビジョニング

以下の手順では、OpenShift Ansible Broker で利用可能にされている PostgreSQL サービスバンドル (APB) のサンプルをプロビジョニングします。

前提条件

- サービスカタログがインストールされていること。

- OpenShift Ansible Broker がインストールされていること。

手順

1. プロジェクトを作成します。
 - a. Web コンソールで、**Home** → **Projects** に移動し、**Create Project** をクリックします。
 - b. **test-postgresql-apb** を **Name** フィールドに入力し、**Create** をクリックします。
2. サービスインスタンスを作成します。
 - a. **Catalog** → **Developer Catalog** ページに移動します。
 - b. **PostgreSQL (APB)** サービスバンドルを選択し、**Create Service Instance** をクリックします。
 - c. デフォルトの選択を確認し、それ以外の必要なフィールドを設定してから **Create** をクリックします。
 - d. **Catalog** → **Provisioned Services** に移動し、**dh-postgresql-apb** サービスインスタンスが作成され、ステータスが **Ready** であることを確認します。
Home → **Events** ページで進捗を確認できます。しばらくすると、**dh-postgresql-apb** のイベントが「The instance was provisioned successfully」というメッセージと共に表示されるはずです。
3. サービスバインディングを作成します。
 - a. **Provisioned Services** ページから、**dh-postgresql-apb** をクリックし、**Create Service Binding** をクリックします。
 - b. デフォルトのサービスバインディング名を確認し、**Create** をクリックします。
これにより、指定された名前を使用してバインディングの新規シークレットが作成されます。
4. 作成されたシークレットを確認します。
 - a. **Workloads** → **Secrets** に移動し、**dh-postgresql-apb** という名前のシークレットが作成されていることを確認します。
 - b. **dh-postgresql-apb** をクリックし、他のアプリへのバインディングに使用されるキーと値のペアを **Data** セクションで確認します。

3.8. OPENSIFT ANSIBLE BROKER のアンインストール

OpenShift Ansible Broker は、これが提供するサービスバンドルへのアクセスがなくなった場合にアンインストールできます。



重要

OpenShift Ansible Broker は OpenShift Container Platform 4 では非推奨になっています。同等または強化された機能は Operator Framework および Operator Lifecycle Manager (OLM) で提供されます。

3.8.1. OpenShift Ansible Broker のアンインストール

以下の手順では、Web コンソールを使用して OpenShift Ansible Broker およびその Operator をアンインストールします。



警告

クラスターに OpenShift Ansible Broker からプロビジョニングされたサービスがある場合には、これをアンインストールしないでください。アンインストールすると、サービスを管理しようとする際にエラーが生じる可能性があります。

前提条件

- OpenShift Ansible Broker がインストールされていること。

手順

この手順では、OpenShift Ansible Broker が **openshift-ansible-service-broker** プロジェクトにインストールされていることを前提とします。

1. OpenShift Ansible Broker をアンインストールします。
 - a. **Operators** → **Installed Operators** に移動し、ドロップダウンメニューから **openshift-ansible-service-broker** プロジェクトを選択します。
 - b. **OpenShift Ansible Service Broker Operator** をクリックします。
 - c. **Automation Broker** タブを選択します。
 - d. **ansible-service-broker** をクリックします。
 - e. **Actions** ドロップダウンメニューから、**Delete Automation Broker** を選択します。
 - f. 確認ポップアップ画面から **Delete** をクリックします。
OpenShift Ansible Broker のアンインストールが終了し、サービスバンドルは Developer Catalog からすぐに削除されます。
2. OpenShift Ansible Service Broker Operator のアンインストール
 - a. **Operators** → **Installed Operators** ページから、**Filter by name** にスクロールするか、キーワードを入力して OpenShift Ansible Service Broker Operator を検索してから、これをクリックします。
 - b. **Operator Details** ページの右側で、**Actions** ドロップダウンメニューから **Uninstall Operator** を選択します。
 - c. **Remove Operator Subscription** ウィンドウでプロンプトが表示されたら、インストールに関連するすべてのコンポーネントを削除する場合は、**Also completely remove the Operator from the selected namespace** チェックボックスをオプションで選択します。これにより CSV が削除され、次に Operator に関連付けられた Pod、Deployment、CRD および CR が削除されます。
 - d. **Remove** を選択します。この Operator は実行を停止し、更新を受信しなくなります。

OpenShift Ansible Service Broker Operator がクラスターにインストールされていない状態になります。

OpenShift Ansible Broker のアンインストール後に、ユーザーは OpenShift Ansible Broker で提供されるサービス バンドルにアクセスできなくなります。

第4章 DEPLOYMENT

4.1. DEPLOYMENT および DEPLOYMENTCONFIG について

OpenShift Container Platform の **Deployment** および **DeploymentConfig** は、一般的なユーザーアプリケーションに対する詳細な管理を行うためのよく似ているものの、異なる2つの方法を提供します。これらは、以下の個別の API オブジェクトで構成されています。

- アプリケーションの特定のコンポーネントの必要な状態を記述する、Pod テンプレートとしての **DeploymentConfig** または **Deployment**。
- **DeploymentConfig** には1つまたは複数の **ReplicationController** が使用され、これには Pod テンプレートとしての **DeploymentConfig** の特定の時点の状態のレコードが含まれます。同様に、**Deployment** には **ReplicationController** を継承する1つ以上の **ReplicaSet** が使用されます。
- アプリケーションの特定バージョンのインスタンスを表す1つ以上の Pod。

4.1.1. デプロイメントのビルディングブロック

Deployment および **DeploymentConfig** は、それぞれビルディングブロックとして、ネイティブ Kubernetes API オブジェクトの **ReplicationController** および **ReplicaSet** の使用によって有効にされます。

ユーザーは、**DeploymentConfig** または **Deployment** によって所有される **ReplicationController**、**ReplicaSet**、または **Pod** を操作する必要はありません。デプロイメントシステムは変更を適切に伝播します。

ヒント

既存のデプロイメントストラテジーが特定のユースケースに適さない場合で、デプロイメントのライフサイクル期間中に複数の手順を手動で実行する必要がある場合は、カスタムデプロイメントストラテジーを作成することを検討してください。

以下のセクションでは、これらのオブジェクトの詳細情報を提供します。

4.1.1.1. ReplicationController

ReplicationController は、Pod の指定された数のレプリカが常時実行されるようにします。Pod が終了するか、または削除される場合、**ReplicationController** 定義された数を満たすように追加のインスタンス化を行います。同様に、必要以上の数の Pod が実行されている場合には、定義された数に一致させるために必要な数の Pod を削除します。

ReplicationController 設定は以下で構成されています。

- 必要なレプリカ数 (これはランタイム時に調整可能)。
- レプリケートされた Pod の作成時に使用する Pod 定義。
- 管理された Pod を特定するためのセレクター。

セレクターは、**ReplicationController** が管理する Pod に割り当てられるラベルセットです。これらのラベルは、Pod 定義に組み込まれ、**ReplicationController** がインスタンス化します。

ReplicationController は、必要に応じて調整できるように、セレクターを使用してすでに実行中の Pod 数

を判別します。

ReplicationController は、負荷またはトラフィックに基づいて自動スケーリングを実行せず、追跡も実行しません。この場合は、レプリカ数を外部の自動スケーラーで調整する必要があります。

以下は ReplicationController 定義の例です。

```
apiVersion: v1
kind: ReplicationController
metadata:
  name: frontend-1
spec:
  replicas: 1 ①
  selector: ②
    name: frontend
  template: ③
    metadata:
      labels: ④
        name: frontend ⑤
    spec:
      containers:
      - image: openshift/hello-openshift
        name: helloworld
      ports:
      - containerPort: 8080
        protocol: TCP
      restartPolicy: Always
```

- ① 実行する Pod のコピー数。
- ② 実行する Pod のラベルセレクター。
- ③ コントローラーが作成する Pod のテンプレート。
- ④ Pod のラベルにはラベルセレクターからのものが含まれます。
- ⑤ パラメーター拡張後の名前の最大長さは 63 文字です。

4.1.1.2. ReplicaSet

ReplicationController と同様に、ReplicaSet は、指定された数の Pod レプリカが特定の時点で実行されるようにするネイティブの Kubernetes API オブジェクトです。ReplicaSet と ReplicationController の相違点は、ReplicaSet ではセットベースのセレクター要件をサポートし、レプリケーションコントローラーは等価ベースのセレクター要件のみをサポートする点です。



注記

カスタム更新のオーケストレーションが必要な場合や、更新が全く必要のない場合にのみ ReplicaSet を使用します。それ以外は Deployment を使用します。ReplicaSet は個別に使用できますが、Pod の作成/削除/更新のオーケストレーションを実行するためにデプロイメントで使用されます。Deployment は ReplicaSet を自動的に管理し、Pod に宣言型の更新を加えるので、作成する ReplicaSet を手動で管理する必要はありません。

以下は、**ReplicaSet** 定義の例になります。

```

apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend-1
  labels:
    tier: frontend
spec:
  replicas: 3
  selector: ❶
    matchLabels: ❷
      tier: frontend
    matchExpressions: ❸
      - {key: tier, operator: In, values: [frontend]}
  template:
    metadata:
      labels:
        tier: frontend
    spec:
      containers:
      - image: openshift/hello-openshift
        name: helloworld
        ports:
        - containerPort: 8080
          protocol: TCP
        restartPolicy: Always

```

- ❶ 一連のリソースに対するラベルのクエリー。**matchLabels** と **matchExpressions** の結果は論理的に結合されます。
- ❷ セレクターに一致するラベルでリソースを指定する等価ベースのセレクター
- ❸ キーをフィルターするセットベースのセレクター。これは、**tier** と同等のキー、**frontend** と同等の値のリソースをすべて選択します。

4.1.2. DeploymentConfig

ReplicationController ベースにビルドされた OpenShift Container Platform は、**DeploymentConfig** の概念に基づいてソフトウェア開発およびデプロイメントライフサイクルの拡張サポートを追加します。最も単純なケースでは、DeploymentConfig は新規の ReplicationController を作成し、これに Pod を起動させます。

ただし、DeploymentConfig の OpenShift Container Platform デプロイメントは、イメージの既存デプロイメントから新規デプロイメントに移行する機能を提供し、ReplicationController の作成前後に実行されるフックも定義します。

DeploymentConfig デプロイメントシステムは以下の機能を提供します。

- アプリケーションを実行するためのテンプレートである DeploymentConfig。
- イベントへの対応として自動化されたデプロイメントを駆動するトリガー。

- 直前のバージョンから新規バージョンに移行するためのユーザーによるカスタマイズが可能なデプロイメントストラテジー。ストラテジーは通常デプロイメントプロセスと呼ばれ、Pod内で実行されます。
- デプロイメントのライフサイクル中の異なる時点でカスタム動作を実行するためのフックのセット (ライフサイクルフック)。
- デプロイメントの失敗時に手動または自動でロールバックをサポートするためのアプリケーションのバージョン管理。
- レプリケーションの手動および自動スケーリング。

DeploymentConfig を作成すると、ReplicationController が、DeploymentConfig の Pod テンプレートとして作成されます。DeploymentConfig が変更されると、最新の Pod テンプレートで新しい ReplicationController が作成され、デプロイメントプロセスが実行されて以前の ReplicationController のスケールダウン、および新規 ReplicationController のスケールアップが行われます。

アプリケーションのインスタンスは、作成時にサービスローダー・バランサーやルーターに対して自動的に追加/削除されます。アプリケーションが正常なシャットダウン機能をサポートしている限り、アプリケーションが **TERM** シグナルを受け取ると、実行中のユーザー接続が通常通り完了できるようになります。

OpenShift Container Platform **DeploymentConfig** オブジェクトは以下の詳細を定義します。

1. **ReplicationController** 定義の要素。
2. 新規デプロイメントの自動作成のトリガー。
3. デプロイメント間の移行ストラテジー。
4. ライフサイクルフック。

デプロイヤー Pod は、デプロイメントがトリガーされるたびに、手動または自動であるかを問わず、(古い ReplicationController のスケールダウン、新規 ReplicationController のスケールアップおよびフックの実行などの) デプロイメントを管理します。デプロイメント Pod は、Deployment のログを維持するために Deployment の完了後は無期限で保持されます。デプロイメントが別のものに置き換えられる場合、以前の ReplicationController は必要に応じて簡単なロールバックを有効にできるように保持されます。

DeploymentConfig 定義の例

```
apiVersion: v1
kind: DeploymentConfig
metadata:
  name: frontend
spec:
  replicas: 5
  selector:
    name: frontend
  template: { ... }
  triggers:
  - type: ConfigChange 1
  - imageChangeParams:
      automatic: true
      containerNames:
      - helloworld
    from:
```



```

kind: ImageStreamTag
name: hello-openshift:latest
type: ImageChange ❷
strategy:
  type: Rolling ❸

```

- ❶ **ConfigChange** トリガーにより、新規 Deployment は ReplicationController テンプレートが変更されるたびに作成されます。
- ❷ **ImageChange** トリガーにより、新規 Deployment は、バッキングイメージの新規バージョンが名前付きイメージストリームで利用可能になるたびに作成されます。
- ❸ デフォルトの **Rolling** ストラテジーにより、Deployment 間のダウンタイムなしの移行が行われま

4.1.3. Deployment

Kubernetes は、**Deployment** という OpenShift Container Platform のファーストクラスのネイティブ API オブジェクトを提供します。Deployment は、OpenShift Container Platform 固有の DeploymentConfig として機能します。

DeploymentConfig の様に、Deployment は Pod テンプレートとして、アプリケーションの特定コンポーネントの必要な状態を記述します。Deployment は、Pod のライフサイクルをオーケストレーションする ReplicaSet を作成します。

たとえば、以下の Deployment 定義は ReplicaSet を作成し、1つの **hello-openshift** Pod を起動します。

Deployment の定義

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: hello-openshift
spec:
  replicas: 1
  selector:
    matchLabels:
      app: hello-openshift
  template:
    metadata:
      labels:
        app: hello-openshift
    spec:
      containers:
        - name: hello-openshift
          image: openshift/hello-openshift:latest
          ports:
            - containerPort: 80

```

4.1.4. Deployment および DeploymentConfig の比較

Kubernetes Deployment および OpenShift Container Platform でプロビジョニングされる DeploymentConfig の両方が OpenShift Container Platform でサポートされていますが、

DeploymentConfig で提供される特定の機能または動作が必要でない場合、Deployment を使用することが推奨されます。

以下のセクションでは、使用するタイプの決定に役立つ 2 つのオブジェクト間の違いを詳述します。

4.1.4.1. 設計

Deployment と DeploymentConfig の重要な違いの 1 つとして、ロールアウトプロセスで各設計で選択される [CAP theorem \(原則\)](#) のプロパティがあります。DeploymentConfig は整合性を優先しますが、Deployment は整合性よりも可用性を優先します。

DeploymentConfig の場合、デプロイ Pod を実行するノードがダウンする場合、ノードの置き換えは行われません。プロセスは、ノードが再びオンラインになるまで待機するか、または手動で削除されます。ノードを手動で削除すると、対応する Pod も削除されます。つまり、kubelet は関連付けられた Pod も削除するため、Pod を削除してロールアウトの固定解除を行うことはできません。

一方、Deployment ロールアウトはコントローラーマネージャーから実行されます。コントローラーマネージャーはマスター上で高可用性モードで実行され、リーダー選択アルゴリズムを使用して可用性を整合性よりも優先するように設定します。障害の発生時には、他の複数のマスターが同時に同じ Deployment に対して作用する可能性があります。この問題は障害の発生直後に調整されます。

4.1.4.2. DeploymentConfig 固有の機能

自動ロールバック

現時点で、Deployment では、問題の発生時の最後に正常にデプロイされた ReplicaSet への自動ロールバックをサポートしていません。

トリガー

Deployment の場合、デプロイメントの Pod テンプレートに変更があるたびに新しいロールアウトが自動的にトリガーされるので、暗黙的な **ConfigChange** トリガーが含まれます。Pod テンプレートの変更時に新たなロールアウトが不要な場合には、デプロイメントを以下のように停止します。

```
$ oc rollout pause deployments/<name>
```

ライフサイクルフック

Deployment ではライフサイクルフックをサポートしていません。

カスタムストラテジー

Deployment では、ユーザーが指定するカスタムデプロイメントストラテジーをサポートしていません。

4.1.4.3. Deployment 固有の機能

ロールオーバー

Deployment のデプロイメントプロセスは、すべての新規ロールアウトにデプロイ Pod を使用する DeploymentConfig とは対照的に、コントローラーループで実行されます。つまり、Deployment は任意の数のアクティブな ReplicaSet を指定することができ、デプロイメントコントローラーがすべての古い ReplicaSet をスケールダウンし、最新の ReplicaSet をスケールアップします。

DeploymentConfig では、実行できるデプロイ Pod は最大 1 つとなっています。複数のデプロイヤーがある場合は競合が生じ、それぞれが最新の ReplicationController であると考えてコントローラーをスケールアップしようとします。これにより、2 つの ReplicationController のみを一度にアクティブにできます。最終的には Deployment のロールアウトが加速します。

比例スケーリング

Deployment コントローラーのみが Deployment が所有する新旧の ReplicaSet のサイズについての信頼できる情報源であるため、継続中のロールアウトのスケールリングが可能です。追加のレプリカはそれぞれの ReplicaSet のサイズに比例して分散されます。

DeploymentConfig については、DeploymentConfig コントローラーが新規 ReplicationController のサイズに関してデプロイヤープロセスと競合するためにロールアウトが継続されている場合はスケールリングできません。

ロールアウト中の一時停止

Deployment はいつでも一時停止できます。つまり、継続中のロールアウトも一時停止できます。一方、デプロイヤー Pod は現時点で一時停止できないので、ロールアウト時に DeploymentConfig を一時停止しようとしても、デプロイヤープロセスはこの影響を受けず、完了するまで続行されます。

4.2. デプロイメントプロセスの管理

4.2.1. DeploymentConfig の管理

DeploymentConfig は、OpenShift Container Platform Web コンソールの **Workloads** ページからか、または **oc** CLI を使用して管理できます。以下の手順は、特に指定がない場合の CLI の使用方法を示しています。

4.2.1.1. デプロイメントの開始

アプリケーションのデプロイメントプロセスを開始するために、**ロールアウト** を開始できます。

手順

1. 既存の DeploymentConfig から新規デプロイメントプロセスを開始するには、以下のコマンドを実行します。

```
$ oc rollout latest dc/<name>
```



注記

デプロイメントプロセスが進行中の場合には、このコマンドを実行すると、メッセージが表示され、新規 ReplicationController はデプロイされません。

4.2.1.2. デプロイメントの表示

アプリケーションの利用可能なすべてのリビジョンについての基本情報を取得するためにデプロイメントを表示できます。

手順

1. 現在実行中のデプロイメントプロセスを含む、指定した DeploymentConfig についての最近作成されたすべての ReplicationController についての詳細を表示するには、以下を実行します。

```
$ oc rollout history dc/<name>
```

2. リビジョンに固有の詳細情報を表示するには、**--revision** フラグを追加します。

```
$ oc rollout history dc/<name> --revision=1
```

3. デプロイメント設定およびその最新バージョンの詳細については、**oc describe** コマンドを使用します。

```
$ oc describe dc <name>
```

4.2.1.3. デプロイメントの再試行

現行リビジョンの DeploymentConfig がデプロイに失敗した場合、デプロイメントプロセスを再起動することができます。

手順

1. 失敗したデプロイメントプロセスを再起動するには、以下を実行します。

```
$ oc rollout retry dc/<name>
```

最新リビジョンのデプロイメントに成功した場合には、このコマンドによりメッセージが表示され、デプロイメントプロセスはこれ以上試行されません。



注記

デプロイメントを再試行すると、デプロイメントプロセスが再起動され、新しいデプロイメントリビジョンは作成されません。再起動された ReplicationController は、失敗したときと同じ設定を使用します。

4.2.1.4. デプロイメントのロールバック

ロールバックすると、アプリケーションを以前のリビジョンに戻します。この操作は、REST API、CLI または Web コンソールで実行できます。

手順

1. 最後にデプロイして成功した設定のリビジョンにロールバックするには以下を実行します。

```
$ oc rollout undo dc/<name>
```

DeploymentConfig のテンプレートは、undo コマンドで指定されたデプロイメントのリビジョンと一致するように元に戻され、新規 ReplicationController が起動します。**--to-revision** でリビジョンが指定されない場合には、最後に成功したデプロイメントのリビジョンが使用されます。

2. ロールバックの完了直後に新規デプロイメントプロセスが誤って開始されないように、DeploymentConfig のイメージ変更トリガーがロールバックの一部として無効にされます。イメージ変更トリガーを再度有効にするには、以下を実行します。

```
$ oc set triggers dc/<name> --auto
```



注記

DeploymentConfig は、最新のデプロイメントプロセスが失敗した場合の、設定の最後に成功したリビジョンへの自動ロールバックもサポートします。この場合、デプロイに失敗した最新のテンプレートはシステムで修正されないため、ユーザーがその設定の修正を行う必要があります。

4.2.1.5. コンテナ内でのコマンドの実行

コマンドをコンテナに追加して、イメージの **ENTRYPOINT** を却下してコンテナの起動動作を変更することができます。これは、指定したタイミングでデプロイメントごとに1回実行できるライフサイクルフックとは異なります。

手順

1. **command** パラメーターを、DeploymentConfig の **spec** フィールドを追加します。 **command** コマンドを変更する **args** フィールドも追加できます (または **command** が存在しない場合には、**ENTRYPOINT**)。

```
spec:
  containers:
  -
    name: <container_name>
    image: 'image'
    command:
    - '<command>'
    args:
    - '<argument_1>'
    - '<argument_2>'
    - '<argument_3>'
```

たとえば、**-jar** および **/opt/app-root/springboots2idemo.jar** 引数を指定して、**java** コマンドを実行するには、以下を実行します。

```
spec:
  containers:
  -
    name: example-spring-boot
    image: 'image'
    command:
    - java
    args:
    - '-jar'
    - '/opt/app-root/springboots2idemo.jar'
```

4.2.1.6. デプロイメントログの表示

手順

1. 指定の DeploymentConfig に関する最新リビジョンのログをストリームするには、以下を実行します。

```
$ oc logs -f dc/<name>
```

最新のリビジョンが実行中または失敗した場合には、コマンドが、Pod のデプロイを行うプロセスのログを返します。成功した場合には、アプリケーションの Pod からのログを返します。

2. 以前に失敗したデプロイメントプロセスからのログを表示することも可能です。ただし、これらのプロセス (以前の ReplicationController およびデプロイヤー Pod) が存在し、手動でプルーニングまたは削除されていない場合に限りです。

```
$ oc logs --version=1 dc/<name>
```

4.2.1.7. デプロイメントトリガー

DeploymentConfig には、クラスター内のイベントに対応する新規デプロイメントプロセスの作成を駆動するトリガーを含めることができます。



警告

トリガーが DeploymentConfig に定義されていない場合は、**ConfigChange** トリガーがデフォルトで追加されます。トリガーが空のフィールドとして定義されている場合には、デプロイメントは手動で起動する必要があります。

ConfigChange デプロイメントトリガー

ConfigChange トリガーにより、DeploymentConfig の Pod テンプレートで設定の変更が検出されるたびに、新規の ReplicationController が作成されます。



注記

ConfigChange トリガーが DeploymentConfig に定義されている場合は、DeploymentConfig 自体が作成された直後に、最初の ReplicationController が自動的に作成され、一時停止されません。

ConfigChange デプロイメントトリガー

```
triggers:
  - type: "ConfigChange"
```

ImageChange デプロイメントトリガー

ImageChange トリガーにより、イメージストリームタグの内容の変更時に常に新規 ReplicationController が生成されます (イメージの新規バージョンがプッシュされるタイミング)。

ImageChange デプロイメントトリガー

```
triggers:
  - type: "ImageChange"
    imageChangeParams:
      automatic: true 1
      from:
        kind: "ImageStreamTag"
        name: "origin-ruby-sample:latest"
        namespace: "myproject"
      containerNames:
        - "helloworld"
```

1 **imageChangeParams.automatic** フィールドが **false** に設定されると、トリガーが無効になります。

上記の例では、**origin-ruby-sample** イメージストリームの **latest** タグの値が変更され、新しいイメージの値が DeploymentConfig の **helloworld** コンテナに指定されている現在のイメージと異なる場合に、**helloworld** コンテナの新規イメージを使用して、新しい ReplicationController が作成されます。



注記

ImageChange トリガーが DeploymentConfig (**ConfigChange** トリガーと **automatic=false**、または **automatic=true**) で定義されていて、**ImageChange** トリガーで参照されている **ImageStreamTag** がまだ存在していない場合には、ビルドにより、イメージが、**ImageStreamTag** にインポートまたはプッシュされた直後に初回のデプロイメントプロセスが自動的に開始されます。

4.2.1.7.1. デプロイメントトリガーの設定

手順

1. **oc set triggers** コマンドを使用して、DeploymentConfig にデプロイメントトリガーを設定することができます。たとえば、**ImageChangeTrigger** を設定するには、以下のコマンドを使用します。

```
$ oc set triggers dc/<dc_name> \
  --from-image=<project>/<image>:<tag> -c <container_name>
```

4.2.1.8. デプロイメントリソースの設定



注記

このリソースはクラスター管理者が一時ストレージのテクノロジープレビュー機能を有効にしている場合にのみ利用できます。この機能はデフォルトでは無効にされています。

デプロイメントは、ノードでリソース (メモリー、CPU および一時ストレージ) を消費する Pod を使用して完了します。デフォルトで、Pod はバインドされていないノードのリソースを消費します。ただし、プロジェクトにデフォルトのコンテナ制限が指定されている場合には、Pod はその上限までリソースを消費します。

デプロイメントストラテジーの一部としてリソース制限を指定して、リソースの使用を制限することも可能です。デプロイメントリソースは、Recreate (再作成)、Rolling (ローリング) または Custom (カスタム) のデプロイメントストラテジーで使用できます。

手順

1. 以下の例では、**resources**、**cpu**、**memory**、および **ephemeral-storage** はそれぞれオプションです。

```
type: "Recreate"
resources:
  limits:
    cpu: "100m" ①
    memory: "256Mi" ②
    ephemeral-storage: "1Gi" ③
```

- 1 **cpu** は CPU のユニットで、**100m** は 0.1 CPU ユニット ($100 * 1e-3$) を表します。
- 2 **memory** はバイト単位です。**256Mi** は 268435456 バイトを表します ($256 * 2^{20}$)。
- 3 **ephemeral-storage** はバイト単位です。**1Gi** は 1073741824 バイト (2^{30}) を表します。この項目は、クラスター管理者が一時ストレージのテクノロジープレビュー機能を有効にしている場合のみ該当します。

ただし、クォータがプロジェクトに定義されている場合には、以下の 2 つの項目のいずれかが必要です。

- 明示的な **requests** で設定した **resources** セクション:

```
type: "Recreate"
resources:
  requests: 1
    cpu: "100m"
    memory: "256Mi"
    ephemeral-storage: "1Gi"
```

- 1 **requests** オブジェクトは、クォータ内のリソース一覧に対応するリソース一覧を含みます。
- プロジェクトで定義される制限の範囲。**LimitRange** オブジェクトのデフォルト値がデプロイメントプロセス時に作成される Pod に適用されます。

デプロイメントリソースを設定するには、上記のいずれかのオプションを選択してください。それ以外の場合は、デプロイ Pod の作成は、クォータ基準を満たしていないことを示すメッセージを出して失敗します。

4.2.1.9. 手動のスケーリング

ロールバック以外に、手動スケーリングにより、レプリカの数を実際に管理できます。



注記

Pod は **oc autoscale** コマンドを使用して自動スケーリングすることも可能です。

手順

1. DeploymentConfig を手動でスケーリングするには、**oc scale** コマンドを使用します。たとえば、以下のコマンドは、**frontend** DeploymentConfig のレプリカを **3** に設定します。

```
$ oc scale dc frontend --replicas=3
```

レプリカ数は最終的に、DeploymentConfig の **frontend** で設定した希望のデプロイメントの状態と現在のデプロイメントの状態に伝播されます。

4.2.1.10. DeploymentConfig からのプライベートリポジトリへのアクセス

シークレットを DeploymentConfig に追加し、プライベートリポジトリからイメージにアクセスできるようにします。この手順では、OpenShift Container Platform Web コンソールを使用する方法を示します。

手順

1. 新規プロジェクトを作成します。
2. **Workloads** ページから、プライベートイメージリポジトリにアクセスするための認証情報を含むシークレットを作成します。
3. DeploymentConfig を作成します。
4. DeploymentConfig エディターページで、**Pull Secret** を設定し、変更を保存します。

4.2.1.11. 特定のノードへの Pod の割り当て

ラベル付きのノードと合わせてノードセクターを使用し、Pod の配置を制御することができます。

クラスター管理者は、プロジェクトに対してデフォルトのノードセクターを設定して特定のノードに Pod の配置を制限できます。開発者は、Pod 設定にノードセクターを設定して、ノードをさらに制限することができます。

手順

1. Pod の作成時にノードセクターを追加するには、Pod 設定を編集し、**nodeSelector** の値を追加します。これは、単一の Pod 設定や、Pod テンプレートに追加できます。

```
apiVersion: v1
kind: Pod
spec:
  nodeSelector:
    disktype: ssd
  ...
```

ノードセクターが有効な場合に作成される Pod は指定されたラベルを持つノードに割り当てられます。ここで指定されるラベルは、クラスター管理者によって追加されるラベルと併用されます。

たとえば、プロジェクトに **type=user-node** と **region=east** のラベルがクラスター管理者により追加され、上記の **disktype: ssd** ラベルを Pod に追加した場合に、Pod は 3 つのラベルすべてが含まれるノードにのみスケジュールされます。



注記

ラベルには値を1つしか設定できないので、**region=east** が管理者によりデフォルト設定されている Pod 設定に **region=west** のノードセクターを設定すると、Pod が全くスケジュールされなくなります。

4.2.1.12. 異なるサービスアカウントでの Pod の実行

デフォルト以外のサービスアカウントで Pod を実行できます。

手順

1. DeploymentConfig を編集します。

```
$ oc edit dc/<deployment_config>
```

2. **serviceAccount** と **serviceAccountName** パラメーターを **spec** フィールドに追加し、使用するサービスアカウントを指定します。

```
spec:
  securityContext: {}
  serviceAccount: <service_account>
  serviceAccountName: <service_account>
```

4.3. DEPLOYMENTCONFIG ストラテジーの使用

デプロイメントストラテジーは、アプリケーションを変更またはアップグレードする1つの方法です。この目的は、ユーザーには改善が加えられていることが分からないように、ダウンタイムなしに変更を加えることにあります。

エンドユーザーは通常ルーターによって処理されるルート経由でアプリケーションにアクセスするため、デプロイメントストラテジーは、DeploymentConfig 機能またはルーティング機能に重点を置きます。DeploymentConfig に重点を置くストラテジーは、アプリケーションを使用するすべてのルートに影響を与えます。ルーター機能を使用するストラテジーは個別のルートにターゲットを設定します。

デプロイメントストラテジーの多くは、DeploymentConfig でサポートされ、追加のストラテジーはルーター機能でサポートされます。このセクションでは、DeploymentConfig ストラテジーについて説明します。

デプロイメントストラテジーの選択

デプロイメントストラテジーを選択する場合に、以下を考慮してください。

- 長期間実行される接続は正しく処理される必要があります。
- データベースの変換は複雑になる可能性があり、アプリケーションと共に変換し、ロールバックする必要があります。
- アプリケーションがマイクロサービスと従来のコンポーネントを使用するハイブリッドの場合には、移行の完了時にダウンタイムが必要になる場合があります。
- これを実行するためのインフラストラクチャーが必要です。
- テスト環境が分離されていない場合は、新規バージョンと以前のバージョン両方が破損してしまう可能性があります。

デプロイメントストラテジーは、readiness チェックを使用して、新しい Pod の使用準備ができているかを判断します。readiness チェックに失敗すると、DeploymentConfig は、タイムアウトするまで Pod の実行を再試行します。デフォルトのタイムアウトは、**10m** で、値は **dc.spec.strategy.*params** の **TimeoutSeconds** で設定します。

4.3.1. ローリングストラテジー

ローリングデプロイメントは、以前のバージョンのアプリケーションインスタンスを、新しいバージョンのアプリケーションインスタンスに徐々に置き換えます。ローリングストラテジーは、DeploymentConfig にストラテジーが指定されていない場合に使用されるデフォルトのデプロイメントストラテジーです。

ローリングデプロイメントは通常、新規 Pod が **readiness check** によって **ready** になるのを待機してから、古いコンポーネントをスケールダウンします。重大な問題が生じる場合、ローリングデプロイメントは中止される場合があります。

ローリングデプロイメントの使用のタイミング

- ダウンタイムを発生させずに、アプリケーションの更新を行う場合
- 以前のコードと新しいコードの同時実行がアプリケーションでサポートされている場合

ローリングデプロイメントとは、以前のバージョンと新しいバージョンのコードを同時に実行するという意味です。これは通常、アプリケーションで N-1 互換性に対応する必要があります。

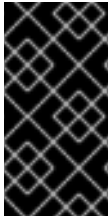
ローリングストラテジ一定義の例

```
strategy:
  type: Rolling
  rollingParams:
    updatePeriodSeconds: 1 ①
    intervalSeconds: 1 ②
    timeoutSeconds: 120 ③
    maxSurge: "20%" ④
    maxUnavailable: "10%" ⑤
    pre: {} ⑥
    post: {}
```

- ① 各 Pod が次に更新されるまで待機する時間。指定されていない場合、デフォルト値は **1** となります。
- ② 更新してからデプロイメントステータスをポーリングするまでの間待機する時間。指定されていない場合、デフォルト値は **1** となります。
- ③ イベントのスケールを中断するまでの待機時間。この値はオプションです。デフォルトは **600** です。ここでの **中断** とは、自動的に以前の完全なデプロイメントにロールバックされるという意味です。
- ④ **maxSurge** はオプションで、指定されていない場合には、デフォルト値は **25%** となります。以下の手順の次にある情報を参照してください。
- ⑤ **maxUnavailable** はオプションで、指定されていない場合には、デフォルト値は **25%** となります。以下の手順の次にある情報を参照してください。
- ⑥ **pre** および **post** はどちらもライフサイクルフックです。

ローリングストラテジー:

1. **pre** ライフサイクルフックを実行します。
2. サージ数に基づいて新しい ReplicationController をスケールアップします。
3. 最大利用不可数に基づいて以前の ReplicationController をスケールダウンします。
4. 新しい ReplicationController が希望のレプリカ数に到達して、以前の ReplicationController の数がゼロになるまで、このスケールを繰り返します。
5. **post** ライフサイクルフックを実行します。



重要

スケールダウン時には、ローリングストラテジーは Pod の準備ができるまで待機し、スケールアップを行うことで可用性に影響が出るかどうかを判断します。Pod をスケールアップしたにもかかわらず、準備が整わない場合には、デプロイメントプロセスは最終的にタイムアウトして、デプロイメントに失敗します。

maxUnavailable パラメーターは、更新時に利用できない Pod の最大数です。**maxSurge** パラメーターは、元の Pod 数を超えてスケジュールできる Pod の最大数です。どちらのパラメーターも、パーセント (例: **10%**) または絶対値 (例: **2**) のいずれかに設定できます。両方のデフォルト値は **25%** です。

以下のパラメーターを使用して、デプロイメントの可用性やスピードを調整できます。以下は例になります。

- **maxUnavailable*=0** および **maxSurge*=20%** が指定されていると、更新時および急速なスケールアップ時に完全なキャパシティが維持されるようになります。
- **maxUnavailable*=10%** および **maxSurge*=0** が指定されていると、追加のキャパシティを使用せずに更新を実行します (インプレース更新)。
- **maxUnavailable*=10%** および **maxSurge*=10%** の場合は、キャパシティが失われる可能性があります。迅速にスケールアップおよびスケールダウンします。

一般的に、迅速にロールアウトする場合は **maxSurge** を使用します。リソースのクォータを考慮して、一部に利用不可の状態が発生してもかまわない場合には、**maxUnavailable** を使用します。

4.3.1.1. カナリアデプロイメント

OpenShift Container Platform におけるすべてのローリングデプロイメントは **カナリアデプロイメント** です。新規バージョン (カナリア) はすべての古いインスタンスが置き換えられる前にテストされます。readiness チェックが成功しない場合、カナリアインスタンスは削除され、DeploymentConfig は自動的にロールバックされます。

readiness チェックはアプリケーションコードの一部であり、新規インスタンスが使用できる状態にするために必要に応じて高度な設定をすることができます。(実際のユーザーワークロードを新規インスタンスに送信するなどの) アプリケーションのより複雑なチェックを実装する必要がある場合、カスタムデプロイメントや blue-green デプロイメントストラテジーの実装を検討してください。

4.3.1.2. ローリングデプロイメントの作成

ローリングデプロイメントは OpenShift Container Platform のデフォルトタイプです。CLI を使用してローリングデプロイメントを作成できます。

手順

1. [DockerHub](#) にあるデプロイメントイメージの例を基にアプリケーションを作成します。

```
$ oc new-app openshift/deployment-example
```

2. ルーターをインストールしている場合は、ルートを使用してアプリケーションを利用できるようにしてください (または、サービス IP を直接使用してください)。

```
$ oc expose svc/deployment-example
```

3. `deployment-example.<project>.<router_domain>` でアプリケーションを参照し、**v1** イメージが表示されることを確認します。
4. レプリカが最大3つになるまで、DeploymentConfig をスケーリングします。

```
$ oc scale dc/deployment-example --replicas=3
```

5. 新しいバージョンの例を **latest** とタグ付けして、新規デプロイメントを自動的にトリガーします。

```
$ oc tag deployment-example:v2 deployment-example:latest
```

6. ブラウザーで、**v2** イメージが表示されるまでページを更新します。
7. CLI を使用している場合は、以下のコマンドで、バージョン1に Pod がいくつあるか、バージョン2にはいくつあるかを表示します。Web コンソールでは、Pod が徐々に v2 に追加され、v1 から削除されます。

```
$ oc describe dc deployment-example
```

デプロイメントプロセス時に、新規 ReplicationController は徐々にスケールアップされます。(readiness チェックをパスした後に) 新規 Pod に **ready** のマークが付けられると、デプロイメントプロセスは継続されます。

Pod が準備状態にならない場合、プロセスは中止し、DeploymentConfig は直前のバージョンにロールバックします。

4.3.2. 再作成ストラテジー

再作成ストラテジーは、基本的なロールアウト動作で、デプロイメントプロセスにコードを挿入するためのライフサイクルフックをサポートします。

再作成ストラテジー定義の例

```
strategy:
  type: Recreate
  recreateParams: ①
  pre: {} ②
  mid: {}
  post: {}
```

- ① **recreateParams** はオプションです。
- ② **pre**、**mid**、および **post** はライフサイクルフックです。

再作成ストラテジー:

1. **pre** ライフサイクルフックを実行します。
2. 以前のデプロイメントをゼロにスケールダウンします。
3. 任意の **mid** ライフサイクルフックを実行します。
4. 新規デプロイメントをスケールアップします。

5. **post** ライフサイクルフックを実行します。



重要

スケールアップ中に、デプロイメントのレプリカ数が複数ある場合は、デプロイメントの最初のレプリカが準備できているかどうかを検証されてから、デプロイメントが完全にスケールアップされます。最初のレプリカの検証に失敗した場合には、デプロイメントは失敗とみなされます。

再作成デプロイメントの使用のタイミング:

- 新規コードを起動する前に、移行または他のデータの変換を行う必要がある場合
- 以前のバージョンと新しいバージョンのアプリケーションコードの同時使用をサポートしていない場合
- 複数のレプリカ間での共有がサポートされていない、RWO ボリュームを使用する場合

再作成デプロイメントでは、短い期間にアプリケーションのインスタンスが実行されなくなるので、ダウンタイムが発生します。ただし、以前のコードと新しいコードは同時には実行されません。

4.3.3. カスタムストラテジー

カスタムストラテジーでは、独自のデプロイメントの動作を提供できるようになります。

カスタムストラテジー定義の例

```
strategy:
  type: Custom
  customParams:
    image: organization/strategy
    command: [ "command", "arg1" ]
  environment:
    - name: ENV_1
      value: VALUE_1
```

上記の例では、**organization/strategy** コンテナイメージにより、デプロイメントの動作が提供されます。オプションの **command** 配列は、イメージの **Dockerfile** で指定した **CMD** ディレクティブを上書きします。指定したオプションの環境変数は、ストラテジープロセスの実行環境に追加されます。

さらに、OpenShift Container Platform は以下の環境変数をデプロイメントプロセスに提供します。

環境変数	説明
OPENSHIFT_DEPLOYMENT_NAME	新規デプロイメント名 (ReplicationController)
OPENSHIFT_DEPLOYMENT_NAMESPACE	新規デプロイメントの namespace

新規デプロイメントのレプリカ数は最初はゼロです。ストラテジーの目的は、ユーザーのニーズに最適な仕方に対応するロジックを使用して新規デプロイメントをアクティブにすることにあります。

または **customParams** を使用して、カスタムのデプロイメントロジックを、既存のデプロイメントストラテジーに挿入します。カスタムのシェルスクリプトロジックを指定して、**openshift-deploy** バイナリーを呼び出します。カスタムのデプロイヤーコンテナイメージを用意する必要はありません。ここでは、代わりにデフォルトの OpenShift Container Platform デプロイヤーイメージが使用されます。

```
strategy:
  type: Rolling
  customParams:
    command:
      - /bin/sh
      - -c
      - |
        set -e
        openshift-deploy --until=50%
        echo Halfway there
        openshift-deploy
        echo Complete
```

この設定により、以下のようなデプロイメントになります。

```
Started deployment #2
--> Scaling up custom-deployment-2 from 0 to 2, scaling down custom-deployment-1 from 2 to 0
(keep 2 pods available, don't exceed 3 pods)
  Scaling custom-deployment-2 up to 1
--> Reached 50% (currently 50%)
Halfway there
--> Scaling up custom-deployment-2 from 1 to 2, scaling down custom-deployment-1 from 2 to 0
(keep 2 pods available, don't exceed 3 pods)
  Scaling custom-deployment-1 down to 1
  Scaling custom-deployment-2 up to 2
  Scaling custom-deployment-1 down to 0
--> Success
Complete
```

カスタムデプロイメントストラテジーのプロセスでは、OpenShift Container Platform API または Kubernetes API へのアクセスが必要な場合には、ストラテジーを実行するコンテナは、認証用のコンテナで利用可能なサービスアカウントのトークンを使用できます。

4.3.4. ライフサイクルフック

ローリングおよび再作成ストラテジーは、ストラテジーで事前に定義したポイントでデプロイメントプロセスに動作を挿入できるようにする **ライフサイクルフック** または **デプロイメントフック** をサポートします。

pre ライフサイクルフックの例

```
pre:
  failurePolicy: Abort
  execNewPod: {} 1
```

1 **execNewPod** は Pod ベースのライフサイクルフックです。

フックにはすべて、フックに問題が発生した場合にストラテジーが取るべきアクションを定義する **failurePolicy** が含まれます。

Abort	フックに失敗すると、デプロイメントプロセスも失敗とみなされます。
Retry	フックの実行は、成功するまで再試行されます。
Ignore	フックの失敗は無視され、デプロイメントは続行されます。

フックには、フックの実行方法を記述するタイプ固有のフィールドがあります。現在、フックタイプとしてサポートされているのは Pod ベースのフックのみで、このフックは **execNewPod** フィールドで指定されます。

Pod ベースのライフサイクルフック

Pod ベースのライフサイクルフックは、DeploymentConfig のテンプレートをベースとする新しい Pod でフックコードを実行します。

以下の DeploymentConfig 例は簡素化されており、この例ではローリングストラテジーを使用します。簡潔にまとめられるように、トリガーおよびその他の詳細は省略しています。

```
kind: DeploymentConfig
apiVersion: v1
metadata:
  name: frontend
spec:
  template:
    metadata:
      labels:
        name: frontend
    spec:
      containers:
        - name: helloworld
          image: openshift/origin-ruby-sample
  replicas: 5
  selector:
    name: frontend
  strategy:
    type: Rolling
    rollingParams:
      pre:
        failurePolicy: Abort
        execNewPod:
          containerName: helloworld ❶
          command: [ "/usr/bin/command", "arg1", "arg2" ] ❷
          env: ❸
            - name: CUSTOM_VAR1
              value: custom_value1
          volumes:
            - data ❹
```

❶ **helloworld** の名前は **spec.template.spec.containers[0].name** を参照します。

❷ この **command** は、**openshift/origin-ruby-sample** イメージで定義される **ENTRYPOINT** を上書きします。


~~~~~

- 3 **env** は、フックコンテナの環境変数です (任意)。
- 4 **volumes** は、フックコンテナのボリューム参照です (任意)。

この例では、**pre** フックは、**helloworld** コンテナからの **openshift/origin-ruby-sample** イメージを使用して新規 Pod で実行されます。フック Pod には以下のプロパティが設定されます。

- フックコマンドは **/usr/bin/command arg1 arg2** です。
- フックコンテナには、**CUSTOM\_VAR1=custom\_value1** 環境変数が含まれます。
- フックの失敗ポリシーは **Abort** で、フックが失敗するとデプロイメントプロセスも失敗します。
- フック Pod は、DeploymentConfig Pod から **data** ボリュームを継承します。

#### 4.3.4.1. ライフサイクルフックの設定

CLI を使用して DeploymentConfig 用に、ライフサイクルフックまたはデプロイメントフックを設定できます。

##### 手順

1. **oc set deployment-hook** コマンドを使用して、必要なフックのタイプを設定します (**--pre**、**--mid**、または **--post**)。たとえば、デプロイメント前のフックを設定するには、以下を実行します。

```
$ oc set deployment-hook dc/frontend \
  --pre -c helloworld -e CUSTOM_VAR1=custom_value1 \
  -v data --failure-policy=abort -- /usr/bin/command arg1 arg2
```

## 4.4. ルートベースのデプロイメントストラテジーの使用

デプロイメントストラテジーは、アプリケーションを進化させる手段として使用します。一部のストラテジーは DeploymentConfigs は、アプリケーションに解決されるすべてのルートのユーザーが確認できる変更を実行します。このセクションで説明される他の高度なストラテジーでは、ルーターを DeploymentConfig と併用して特定のルートに影響を与えます。

最も一般的なルートベースのストラテジーとして **blue-green デプロイメント** を使用します。新規バージョン (blue バージョン) を、テストと評価用に起動しつつ、安定版 (green バージョン) をユーザーが継続して使用します。準備が整ったら、blue バージョンに切り替えられます。問題が発生した場合には、green バージョンに戻すことができます。

一般的な別のストラテジーとして、**A/B バージョン** がいずれも、同時にアクティブな状態で、A バージョンを使用するユーザーも、B バージョンを使用するユーザーもいるという方法があります。これは、ユーザーインターフェースや他の機能の変更をテストして、ユーザーのフィードバックを取得するために使用できます。また、ユーザーに対する問題の影響が限られている場合に、実稼働のコンテキストで操作が正しく行われていることを検証するのに使用することもできます。

カナリアデプロイメントでは、新規バージョンをテストしますが、問題が検出されると、すぐに以前のバージョンにフォールバックされます。これは、上記のストラテジーどちらでも実行できます。

ルートベースのデプロイメントストラテジーでは、サービス内の Pod 数はスケーリングされません。希望とするパフォーマンスの特徴を維持するには、デプロイメント設定をスケーリングする必要がある場合があります。

#### 4.4.1. プロキシシャーードおよびトラフィック分割

実稼働環境で、特定のシャーードに到達するトラフィックの分散を正確に制御できます。多くのインスタンスを扱う場合は、各シャーードに相対的なスケールを使用して、割合ベースのトラフィックを実装できます。これは、他の場所で実行中の別のサービスやアプリケーションに転送または分割する **プロキシシャーード** とも適切に統合されます。

最も単純な設定では、プロキシは要求を変更せずに転送します。より複雑な設定では、受信要求を複製して、別のクラスターだけでなく、アプリケーションのローカルインスタンスにも送信して、結果を比較することができます。他のパターンとしては、DR のインストールのキャッシュを保持したり、分析目的で受信トラフィックをサンプリングすることができます。

TCP (または UDP) のプロキシは必要なシャーードで実行できます。**oc scale** コマンドを使用して、プロキシシャーードで要求に対応するインスタンスの相対数を変更してください。より複雑なトラフィックを管理する場合は、OpenShift Container Platform ルーターを比例分散機能でカスタマイズすることを検討してください。

#### 4.4.2. N-1 互換性

新規コードと以前のコードが同時に実行されるアプリケーションの場合は、新規コードで記述されたデータが、以前のバージョンのコードで読み込みや処理 (または正常に無視) できるように注意する必要があります。これは、**スキーマの進化**と呼ばれる複雑な問題です。

これは、ディスクに保存したデータ、データベース、一時的なキャッシュ、ユーザーのブラウザーセッションの一部など、多数の形式を取ることができます。多くの Web アプリケーションはローリングデプロイメントをサポートできますが、アプリケーションをテストし、設計してこれに対応させることが重要です。

アプリケーションによっては、新旧のコードが並行的に実行されている期間が短いため、バグやユーザーのトランザクションに失敗しても許容範囲である場合があります。別のアプリケーションでは失敗したパターンが原因で、アプリケーション全体が機能しなくなる場合もあります。

N-1 互換性を検証する 1 つの方法として、A/B デプロイメントを使用できます。制御されたテスト環境で、以前のコードと新しいコードを同時に実行して、新規デプロイメントに流れるトラフィックが以前のデプロイメントで問題を発生させないかを確認します。

#### 4.4.3. 正常な終了

OpenShift Container Platform および Kubernetes は、負荷分散のローテーションから削除する前にアプリケーションインスタンスがシャットダウンする時間を設定します。ただし、アプリケーションでは、終了前にユーザー接続が正常に中断されていることを確認する必要があります。

シャットダウン時に、OpenShift Container Platform はコンテナのプロセスに **TERM** シグナルを送信します。**SIGTERM** を受信すると、アプリケーションコードは、新規接続の受け入れを停止します。これにより、ロードバランサーによって他のアクティブなインスタンスにトラフィックがルーティングされるようになります。アプリケーションコードは、開放されている接続がすべて終了する (または、次の機会に個別接続が正常に終了される) まで待機してから終了します。

正常に終了する期間が終わると、終了されていないプロセスに **KILL** シグナルが送信され、プロセスが即座に終了されます。Pod の **terminationGracePeriodSeconds** 属性または Pod テンプレートは正常に終了する期間 (デフォルトの 30 秒) を制御し、必要に応じてこれらをアプリケーションごとにカスタマイズすることができます。

#### 4.4.4. Blue-Green デプロイメント

Blue-green デプロイメントでは、同時にアプリケーションの2つのバージョンを実行し、実稼働版 (green バージョン) からより新しいバージョン (blue バージョン) にトラフィックを移動します。ルートでは、ローリングストラテジーまたは切り替えサービスを使用できます。

多くのアプリケーションは永続データに依存するので、**N-1 互換性**をサポートするアプリケーションが必要です。つまり、データを共有して、データ層を2つ作成し、データベース、ストアまたはディスク間のライブマイグレーションを実装します。

新規バージョンのテストに使用するデータについて考えてみてください。実稼働データの場合には、新規バージョンのバグにより、実稼働版を破損してしまう可能性があります。

##### 4.4.4.1. Blue-Green デプロイメントの設定

Blue-green デプロイメントでは2つの DeploymentConfig を使用します。どちらも実行され、実稼働のデプロイメントはルートが指定するサービスによって変わります。この際、各 DeploymentConfig は異なるサービスに公開されます。



#### 注記

ルートは、Web (HTTP および HTTPS) トラフィックを対象としているので、この手法は Web アプリケーションに最適です。

新規バージョンに新規ルートを作成し、これをテストすることができます。準備ができれば、実稼働ルートのサービスが新規サービスを参照するように変更します。新規 (blue) バージョンは有効になります。

必要に応じて以前のバージョンにサービスを切り替えて、以前の green バージョンにロールバックすることができます。

#### 手順

1. アプリケーションサンプルの2つのコピーを作成します。

```
$ oc new-app openshift/deployment-example:v1 --name=example-green
$ oc new-app openshift/deployment-example:v2 --name=example-blue
```

上記のコマンドにより、独立したアプリケーションコンポーネントが2つ作成されます。1つは、**example-green** サービスで **v1** イメージを実行するコンポーネントと、もう1つは **example-blue** サービスで **v2** イメージを実行するコンポーネントです。

2. 以前のサービスを参照するルートを作成します。

```
$ oc expose svc/example-green --name=bluegreen-example
```

3. **example-green.<project>.<router\_domain>** でアプリケーションを参照し、**v1** イメージが表示されることを確認します。
4. ルートを編集して、サービス名を **example-blue** に変更します。

```
$ oc patch route/bluegreen-example -p '{"spec":{"to":{"name":"example-blue"}}}'
```

5. ルートが変更されたことを確認するには、**v2** イメージが表示されるまで、ブラウザを更新します。

#### 4.4.5. A/B デプロイメント

A/B デプロイメントストラテジーでは、新しいバージョンのアプリケーションを実稼働環境での制限された方法で試すことができます。実稼働バージョンは、ユーザーの要求の大半に対応し、要求の一部が新しいバージョンに移動されるように指定できます。

各バージョンへの要求の割合を制御できるので、テストが進むにつれ、新しいバージョンへの要求を増やし、最終的に以前のバージョンの使用を停止することができます。各バージョン要求負荷を調整する際に、期待どおりのパフォーマンスを出せるように、各サービスの Pod 数もスケーリングする必要があります。

ソフトウェアのアップグレードに加え、この機能を使用してユーザーインターフェースのバージョンを検証することができます。以前のバージョンを使用するユーザーと、新しいバージョンを使用するユーザーが出てくるので、異なるバージョンに対するユーザーの反応を評価して、設計上の意思決定を知らせることができます。

このデプロイメントを有効にするには、以前のバージョンと新しいバージョンは同時に実行できるほど類似している必要があります。これは、バグ修正リリースや新機能が以前の機能と干渉しないようにする場合の一般的なポイントになります。これらのバージョンが正しく連携するには N-1 互換性が必要です。

OpenShift Container Platform は、Web コンソールと CLI で N-1 互換性をサポートします。

##### 4.4.5.1. A/B テスト用の負荷分散

ユーザーは複数のサービスでルートを設定します。各サービスは、アプリケーションの1つのバージョンを処理します。

各サービスには **weight** が割り当てられ、各サービスへの要求の部分については **service\_weight** を **sum\_of\_weights** で除算します。エンドポイントの **weights** の合計がサービスの **weight** になるように、サービスごとの **weight** がサービスのエンドポイントに分散されます。

ルートにはサービスを最大で 4 つ含めることができます。サービスの **weight** は、**0** から **256** の間で指定してください。**weight** が **0** の場合は、サービスはロードバランシングに参加せず、既存の持続する接続を継続的に提供します。サービスの **weight** が **0** でない場合は、エンドポイントの最小 **weight** は **1** となります。これにより、エンドポイントが多数含まれるサービスでは、最終的に **weight** は必要な値よりも大きくなる可能性があります。このような場合は、負荷分散の **weight** を必要なレベルに下げするために Pod の数を減らします。

#### 手順

A/B 環境を設定するには、以下を実行します。

1. 2 つのアプリケーションを作成して、異なる名前を指定します。それぞれが DeploymentConfig を作成します。これらのアプリケーションは同じアプリケーションのバージョンであり、通常 1 つは現在の実稼働バージョンで、もう 1 つは提案される新規バージョンとなります。

```
$ oc new-app openshift/deployment-example --name=ab-example-a
$ oc new-app openshift/deployment-example --name=ab-example-b
```

どちらのアプリケーションもデプロイされ、サービスが作成されます。

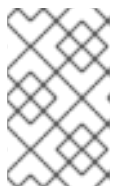
2. ルート経田でアプリケーションを外部から利用できるようにします。この時点でサービスを公開できます。現在の実稼働バージョンを公開してから、後でルート編集して新規バージョンを追加すると便利です。

```
$ oc expose svc/ab-example-a
```

**ab-example-<project>.<router\_domain>** でアプリケーションを参照して、必要なバージョンが表示されていることを確認します。

3. ルートをデプロイする場合には、ルーターはサービスに指定した **weights** に従ってトラフィックを分散します。この時点では、デフォルトの **weight=1** と指定されたサービスが1つ存在するので、すべての要求がこのサービスに送られます。他のサービスを **alternateBackends** として追加し、**weights** を調整すると、A/B 設定が機能するようになります。これは、**oc set route-backends** コマンドを実行するか、ルートを編集して実行できます。

**oc set route-backend** を **0** に設定することは、サービスがロードバランシングに参加しないが、既存の持続する接続を提供し続けることを意味します。



### 注記

ルートに変更を加えると、さまざまなサービスへのトラフィックの部分だけが変更されます。DeploymentConfig をスケーリングして、必要な負荷を処理できるように Pod 数を調整する必要がある場合があります。

ルートを編集するには、以下を実行します。

```
$ oc edit route <route_name>
...
metadata:
  name: route-alternate-service
  annotations:
    haproxy.router.openshift.io/balance: roundrobin
spec:
  host: ab-example.my-project.my-domain
  to:
    kind: Service
    name: ab-example-a
    weight: 10
  alternateBackends:
  - kind: Service
    name: ab-example-b
    weight: 15
...
```

#### 4.4.5.1.1. Web コンソールを使用した重みの管理

##### 手順

1. Route の詳細ページ (Applications/Routes) に移動します。
2. Actions メニューから **Edit** を選択します。
3. **Split traffic across multiple services** にチェックを入れます。
4. **Service Weights** スライダーで、各サービスに送信するトラフィックの割合を設定します。

3つ以上のサービスにトラフィックを分割する場合には、各サービスに0から256の整数を使用して、相対的な重みを指定します。

トラフィックの重みは、トラフィックを分割したアプリケーションの行を展開すると **Overview** に表示されます。

#### 4.4.5.1.2. CLI を使用した重みの管理

##### 手順

1. サービスおよび対応する重みのルートによる負荷分散を管理するには、**oc set route-backends** コマンドを使用します。

```
$ oc set route-backends ROUTENAME \
  [--zero|--equal] [--adjust] SERVICE=WEIGHT[%] [...] [options]
```

たとえば、以下のコマンドは **ab-example-a** に **weight=198** を指定して主要なサービスとし、**ab-example-b** に **weight=2** を指定して1番目の代用サービスとして設定します。

```
$ oc set route-backends ab-example ab-example-a=198 ab-example-b=2
```

つまり、99%のトラフィックはサービス **ab-example-a** に、1%はサービス **ab-example-b** に送信されます。

このコマンドでは、DeploymentConfig はスケーリングされません。要求の負荷を処理するのに十分な Pod がある状態でこれを実行する必要があります。

2. フラグなしのコマンドを実行して、現在の設定を確認します。

```
$ oc set route-backends ab-example
NAME          KIND  TO          WEIGHT
routes/ab-example  Service  ab-example-a 198 (99%)
routes/ab-example  Service  ab-example-b  2  (1%)
```

3. **--adjust** フラグを使用すると、個別のサービスの重みを、それ自体に対して、または主要なサービスに対して相対的に変更できます。割合を指定すると、主要サービスまたは1番目の代用サービス (主要サービスを設定している場合) に対して相対的にサービスを調整できます。他にバックエンドがある場合には、重みは変更按比例した状態になります。以下は例になります。

```
$ oc set route-backends ab-example --adjust ab-example-a=200 ab-example-b=10
$ oc set route-backends ab-example --adjust ab-example-b=5%
$ oc set route-backends ab-example --adjust ab-example-b=+15%
```

**--equal** フラグでは、全サービスの **weight** が **100** になるように設定します。

```
$ oc set route-backends ab-example --equal
```

**--zero** フラグは、全サービスの **weight** を **0** に設定します。すべての要求に対して 503 エラーが返されます。



## 注記

ルートによっては、複数のバックエンドまたは重みが設定されたバックエンドをサポートしないものがあります。

### 4.4.5.1.3.1 サービス、複数の DeploymentConfig

#### 手順

1. すべてのシャードに共通の **ab-example=true** ラベルを追加して新規アプリケーションを作成します。

```
$ oc new-app openshift/deployment-example --name=ab-example-a
```

アプリケーションがデプロイされ、サービスが作成されます。これは最初のシャードです。

2. ルートを使用してアプリケーションを利用できるようにしてください (または、サービス IP を直接使用してください)。

```
$ oc expose svc/ab-example-a --name=ab-example
```

3. **ab-example-<project>.<router\_domain>** でアプリケーションを参照し、**v1** イメージが表示されることを確認します。
4. 1つ目のシャードと同じソースイメージおよびラベルに基づくが、別のバージョンがタグ付けされたバージョンと一意の環境変数を指定して2つ目のシャードを作成します。

```
$ oc new-app openshift/deployment-example:v2 \
  --name=ab-example-b --labels=ab-example=true \
  SUBTITLE="shard B" COLOR="red"
```

5. この時点で、いずれの Pod もルートでサービスが提供されます。しかし、両ブラウザ (接続を開放) とルーター (デフォルトでは cookie を使用) で、バックエンドサーバーへの接続を維持しようとするので、シャードが両方返されない可能性があります。  
1つのまたは他のシャードに対してブラウザを強制的に実行するには、以下を実行します。

- a. **oc scale** コマンドを使用して、**ab-example-a** のレプリカを **0** に減らします。

```
$ oc scale dc/ab-example-a --replicas=0
```

ブラウザを更新して、**v2** および **shard B** (赤) を表示させます。

- b. **ab-example-a** を 1 レプリカに、**ab-example-b** を **0** にスケーリングします。

```
$ oc scale dc/ab-example-a --replicas=1; oc scale dc/ab-example-b --replicas=0
```

ブラウザを更新して、**v1** および **shard A** (青) を表示します。

6. いずれかのシャードでデプロイメントをトリガーする場合、そのシャードの Pod のみが影響を受けます。どちらかの DeploymentConfig で **SUBTITLE** 環境変数を変更してデプロイメントをトリガーできます。

```
$ oc edit dc/ab-example-a
```

または、以下を実行します。

```
┆ $ oc edit dc/ab-example-b
```



## 第5章 クォータ

### 5.1. プロジェクトごとのリソースクォータ

ResourceQuota オブジェクトで定義される **リソースクォータ** は、プロジェクトごとにリソース消費量の総計を制限する制約を指定します。これは、タイプ別にプロジェクトで作成できるオブジェクトの数を制限すると共に、そのプロジェクトのリソースが消費できるコンピュートリソースおよびストレージの合計量を制限することができます。

本書では、リソースクォータの仕組みや、クラスター管理者がリソースクォータはプロジェクトごとにどのように設定し、管理できるか、および開発者やクラスター管理者がそれらをどのように表示できるかについて説明します。

#### 5.1.1. クォータで管理されるリソース

以下では、クォータで管理できる一連のコンピュートリソースとオブジェクトタイプについて説明します。



#### 注記

**status.phase in (Failed, Succeeded)** が true の場合、Pod は終了状態にあります。

表5.1 クォータで管理されるコンピュートリソース

| リソース名             | 説明                                                                                                                                                                                                                    |
|-------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| cpu               | 非終了状態のすべての Pod での CPU 要求の合計はこの値を超えることができません。 <b>cpu</b> および <b>requests.cpu</b> は同じ値で、交換可能なものとして使用できます。                                                                                                               |
| memory            | 非終了状態のすべての Pod でのメモリー要求の合計はこの値を超えることができません <b>memory</b> および <b>requests.memory</b> は同じ値で、交換可能なものとして使用できます。                                                                                                           |
| ephemeral-storage | 非終了状態のすべての Pod でのローカルの一時ストレージ要求の合計は、この値を超えることができません。 <b>ephemeral-storage</b> および <b>requests.ephemeral-storage</b> は同じ値であり、交換可能なものとして使用できます。このリソースは、一時ストレージのテクノロジープレビュー機能が有効にされている場合にのみ利用できます。この機能はデフォルトでは無効にされています。 |
| requests.cpu      | 非終了状態のすべての Pod での CPU 要求の合計はこの値を超えることができません。 <b>cpu</b> および <b>requests.cpu</b> は同じ値で、交換可能なものとして使用できます。                                                                                                               |
| requests.memory   | 非終了状態のすべての Pod でのメモリー要求の合計はこの値を超えることができません <b>memory</b> および <b>requests.memory</b> は同じ値で、交換可能なものとして使用できます。                                                                                                           |

| リソース名                             | 説明                                                                                                                                                                                                               |
|-----------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>requests.ephemeral-storage</b> | 非終了状態のすべての Pod における一時ストレージ要求の合計は、この値を超えることができません。 <b>ephemeral-storage</b> および <b>requests.ephemeral-storage</b> は同じ値で、交換可能なものとして使用できます。このリソースは、一時ストレージのテクノロジープレビュー機能が有効にされている場合にのみ利用できます。この機能はデフォルトでは無効にされています。 |
| <b>limits.cpu</b>                 | 非終了状態のすべての Pod での CPU 制限の合計はこの値を超えることができません。                                                                                                                                                                     |
| <b>limits.memory</b>              | 非終了状態のすべての Pod でのメモリー制限の合計はこの値を超えることができません。                                                                                                                                                                      |
| <b>limits.ephemeral-storage</b>   | 非終了状態のすべての Pod における一時ストレージ制限の合計は、この値を超えることができません。このリソースは、一時ストレージのテクノロジープレビュー機能が有効にされている場合にのみ利用できます。この機能はデフォルトでは無効にされています。                                                                                        |

表5.2 クォータで管理されるストレージリソース

| リソース名                                                                                | 説明                                                                                               |
|--------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------|
| <b>requests.storage</b>                                                              | 任意の状態のすべての Persistent Volume Claim (永続ボリューム要求、PVC) でのストレージ要求の合計は、この値を超えることができません。                |
| <b>persistentvolumeclaims</b>                                                        | プロジェクトに存在できる Persistent Volume Claim (永続ボリューム要求、PVC) の合計数です。                                     |
| <b>&lt;storage-class-name&gt;.storageclass.storage.k8s.io/requests.storage</b>       | 一致するストレージクラスを持つ、任意の状態のすべての Persistent Volume Claim (永続ボリューム要求、PVC) でのストレージ要求の合計はこの値を超えることができません。 |
| <b>&lt;storage-class-name&gt;.storageclass.storage.k8s.io/persistentvolumeclaims</b> | プロジェクトに存在できる、一致するストレージクラスを持つ Persistent Volume Claim (永続ボリューム要求、PVC) の合計数です。                     |

表5.3 クォータで管理されるオブジェクト数

| リソース名                         | 説明                                         |
|-------------------------------|--------------------------------------------|
| <b>pods</b>                   | プロジェクトに存在できる非終了状態の Pod の合計数です。             |
| <b>replicationcontrollers</b> | プロジェクトに存在できる ReplicationController の合計数です。 |

| リソース名                            | 説明                                                           |
|----------------------------------|--------------------------------------------------------------|
| <b>resourcequotas</b>            | プロジェクトに存在できるリソースクォータの合計数です。                                  |
| <b>services</b>                  | プロジェクトに存在できるサービスの合計数です。                                      |
| <b>services.loadbalancers</b>    | プロジェクトに存在できるタイプ <b>LoadBalancer</b> のサービスの合計数です。             |
| <b>services.nodeports</b>        | プロジェクトに存在できるタイプ <b>NodePort</b> のサービスの合計数です。                 |
| <b>secrets</b>                   | プロジェクトに存在できるシークレットの合計数です。                                    |
| <b>configmaps</b>                | プロジェクトに存在できる <b>ConfigMap</b> オブジェクトの合計数です。                  |
| <b>persistentvolumeclaims</b>    | プロジェクトに存在できる Persistent Volume Claim (永続ボリューム要求、PVC) の合計数です。 |
| <b>openshift.io/imagestreams</b> | プロジェクトに存在できるイメージストリームの合計数です。                                 |

### 5.1.2. クォータのスコープ

各クォータには **スコープ** のセットが関連付けられます。クォータは、列挙されたスコープの交差部分に一致する場合にのみリソースの使用状況を測定します。

スコープをクォータに追加すると、クォータが適用されるリソースのセットを制限できます。許可されるセット以外のリソースを設定すると、検証エラーが発生します。

| スコープ                  | 説明                                                                                                |
|-----------------------|---------------------------------------------------------------------------------------------------|
| <b>Terminating</b>    | <b>spec.activeDeadlineSeconds</b> $\geq 0$ の Pod に一致します。                                          |
| <b>NotTerminating</b> | <b>spec.activeDeadlineSeconds</b> が <b>nil</b> の Pod に一致します。                                      |
| <b>BestEffort</b>     | <b>cpu</b> または <b>memory</b> のいずれかについてのサービスの QoS (Quality of Service) が Best Effort の Pod に一致します。 |
| <b>NotBestEffort</b>  | <b>cpu</b> および <b>memory</b> についてのサービスの QoS (Quality of Service) が Best Effort ではない Pod に一致します。   |

**BestEffort** スコープは、以下のリソースに制限するようにクォータを制限します。

- **Pods**

**Terminating**、**NotTerminating**、および **NotBestEffort** スコープは、以下のリソースを追跡するようにクォータを制限します。

- **pods**
- **memory**
- **requests.memory**
- **limits.memory**
- **cpu**
- **requests.cpu**
- **limits.cpu**
- **ephemeral-storage**
- **requests.ephemeral-storage**
- **limits.ephemeral-storage**



#### 注記

一時ストレージ要求と制限は、テクノロジープレビューとして提供されている一時ストレージを有効にした場合にのみ適用されます。この機能はデフォルトでは無効にされています。

### 5.1.3. クォータの実施

プロジェクトのリソースクォータが最初に作成されると、プロジェクトは、更新された使用状況の統計が計算されるまでクォータ制約の違反を引き起こす可能性のある新規リソースの作成機能を制限します。

クォータが作成され、使用状況の統計が更新されると、プロジェクトは新規コンテンツの作成を許可します。リソースを作成または変更する場合、クォータの使用量はリソースの作成または変更要求があるとすぐに増分します。

リソースを削除する場合、クォータの使用量は、プロジェクトのクォータ統計の次の完全な再計算時に減分されます。設定可能な時間を指定して、クォータ使用量の統計値を現在確認されるシステム値まで下げるのに必要な時間を決定します。

プロジェクト変更がクォータ使用制限を超える場合、サーバーはそのアクションを拒否し、クォータ制約を違反していること、およびシステムで現在確認される使用量の統計値を示す適切なエラーメッセージがユーザーに返されます。

### 5.1.4. 要求 vs 制限

コンピュートリソースの割り当て時に、各コンテナは CPU、メモリー、一時ストレージのそれぞれに要求値と制限値を指定できます。クォータはこれらの値のいずれも制限できます。

クォータに **requests.cpu** または **requests.memory** の値が指定されている場合、すべての着信コンテナがそれらのリソースを明示的に要求することが求められます。クォータに **limits.cpu** または **limits.memory** の値が指定されている場合、すべての着信コンテナがそれらのリソースの明示的な制限を指定することが求められます。

### 5.1.5. リソースクォータ定義の例

#### core-object-counts.yaml

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: core-object-counts
spec:
  hard:
    configmaps: "10" ①
    persistentvolumeclaims: "4" ②
    replicationcontrollers: "20" ③
    secrets: "10" ④
    services: "10" ⑤
    services.loadbalancers: "2" ⑥
```

- ① プロジェクトに存在できる **ConfigMap** オブジェクトの合計数です。
- ② プロジェクトに存在できる Persistent Volume Claim (永続ボリューム要求、PVC) の合計数です。
- ③ プロジェクトに存在できる ReplicationController の合計数です。
- ④ プロジェクトに存在できるシークレットの合計数です。
- ⑤ プロジェクトに存在できるサービスの合計数です。
- ⑥ プロジェクトに存在できるタイプ **LoadBalancer** のサービスの合計数です。

#### openshift-object-counts.yaml

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: openshift-object-counts
spec:
  hard:
    openshift.io/imagestreams: "10" ①
```

- ① プロジェクトに存在できるイメージストリームの合計数です。

#### compute-resources.yaml

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: compute-resources
spec:
  hard:
    pods: "4" ①
    requests.cpu: "1" ②
```

```

requests.memory: 1Gi ③
requests.ephemeral-storage: 2Gi ④
limits.cpu: "2" ⑤
limits.memory: 2Gi ⑥
limits.ephemeral-storage: 4Gi ⑦

```

- ① プロジェクトに存在できる非終了状態の Pod の合計数です。
- ② 非終了状態のすべての Pod において、CPU 要求の合計は 1 コアを超えることができません。
- ③ 非終了状態のすべての Pod において、メモリー要求の合計は 1 Gi を超えることができません。
- ④ 非終了状態のすべての Pod において、一時ストレージ要求の合計は 2 Gi を超えることができません。
- ⑤ 非終了状態のすべての Pod において、CPU 制限の合計は 2 コアを超えることができません。
- ⑥ 非終了状態のすべての Pod において、メモリー制限の合計は 2 Gi を超えることができません。
- ⑦ 非終了状態のすべての Pod において、一時ストレージ制限の合計は 4 Gi を超えることができません。

### besteffort.yaml

```

apiVersion: v1
kind: ResourceQuota
metadata:
  name: besteffort
spec:
  hard:
    pods: "1" ①
  scopes:
    - BestEffort ②

```

- ① プロジェクトに存在できるサービスの QoS (Quality of Service) が **BestEffort** の非終了状態の Pod の合計数です。
- ② クォータを、メモリーまたは CPU のいずれかのサービスの QoS (Quality of Service) が **BestEffort** の一致する Pod のみに制限します。

### compute-resources-long-running.yaml

```

apiVersion: v1
kind: ResourceQuota
metadata:
  name: compute-resources-long-running
spec:
  hard:
    pods: "4" ①
    limits.cpu: "4" ②
    limits.memory: "2Gi" ③

```

```
limits.ephemeral-storage: "4Gi" ④
scopes:
- NotTerminating ⑤
```

- ① 非終了状態の Pod の合計数です。
- ② 非終了状態のすべての Pod において、CPU 制限の合計はこの値を超えることができません。
- ③ 非終了状態のすべての Pod において、メモリー制限の合計はこの値を超えることができません。
- ④ 非終了状態のすべての Pod において、一時ストレージ制限の合計はこの値を超えることができません。
- ⑤ クォータを **spec.activeDeadlineSeconds** が **nil** に設定されている一致する Pod のみに制限します。ビルド Pod は、**RestartNever** ポリシーが適用されない場合に **NotTerminating** になります。

### compute-resources-time-bound.yaml

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: compute-resources-time-bound
spec:
  hard:
    pods: "2" ①
    limits.cpu: "1" ②
    limits.memory: "1Gi" ③
    limits.ephemeral-storage: "1Gi" ④
  scopes:
  - Terminating ⑤
```

- ① 非終了状態の Pod の合計数です。
- ② 非終了状態のすべての Pod において、CPU 制限の合計はこの値を超えることができません。
- ③ 非終了状態のすべての Pod において、メモリー制限の合計はこの値を超えることができません。
- ④ 非終了状態のすべての Pod において、一時ストレージ制限の合計はこの値を超えることができません。
- ⑤ クォータを **spec.activeDeadlineSeconds >=0** に設定されている一致する Pod のみに制限します。たとえば、このクォータはビルド Pod またはデプロイヤー Pod に影響を与えますが、web サーバーまたはデータベースなどの長時間実行されない Pod には影響を与えません。

### storage-consumption.yaml

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: storage-consumption
spec:
  hard:
    persistentvolumeclaims: "10" ①
```

```

requests.storage: "50Gi" 2
gold.storageclass.storage.k8s.io/requests.storage: "10Gi" 3
silver.storageclass.storage.k8s.io/requests.storage: "20Gi" 4
silver.storageclass.storage.k8s.io/persistentvolumeclaims: "5" 5
bronze.storageclass.storage.k8s.io/requests.storage: "0" 6
bronze.storageclass.storage.k8s.io/persistentvolumeclaims: "0" 7

```

- 1 プロジェクト内の Persistent Volume Claim (永続ボリューム要求、PVC) の合計数です。
- 2 プロジェクトのすべての Persistent Volume Claim (永続ボリューム要求、PVC) において、要求されるストレージの合計はこの値を超えることができません。
- 3 プロジェクトのすべての Persistent Volume Claim (永続ボリューム要求、PVC) において、gold ストレージクラスで要求されるストレージの合計はこの値を超えることができません。
- 4 プロジェクトのすべての Persistent Volume Claim (永続ボリューム要求、PVC) において、silver ストレージクラスで要求されるストレージの合計はこの値を超えることができません。
- 5 プロジェクトのすべての Persistent Volume Claim (永続ボリューム要求、PVC) において、silver ストレージクラスの要求の合計数はこの値を超えることができません。
- 6 プロジェクトのすべての Persistent Volume Claim (永続ボリューム要求、PVC) において、bronze ストレージクラスで要求されるストレージの合計はこの値を超えることができません。これが 0 に設定される場合、bronze ストレージクラスはストレージを要求できないことを意味します。
- 7 プロジェクトのすべての Persistent Volume Claim (永続ボリューム要求、PVC) において、bronze ストレージクラスで要求されるストレージの合計はこの値を超えることができません。これが 0 に設定される場合は、bronze ストレージクラスでは要求を作成できないことを意味します。

### 5.1.6. クォータの作成

特定のプロジェクトでリソースの使用を制限するためにクォータを作成することができます。

#### 手順

1. ファイルにクォータを定義します。
2. クォータを作成し、これをプロジェクトに適用するためにファイルを使用します。

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

以下は例になります。

```
$ oc create -f core-object-counts.yaml -n demoproject
```

#### 5.1.6.1. オブジェクトカウントクォータの作成

**BuildConfig** および **DeploymentConfig** などの、OpenShift Container Platform の標準的な namespace を使用しているリソースタイプのすべてにオブジェクトカウントクォータを作成できます。オブジェクトクォータカウントは、定義されたクォータをすべての標準的な namespace を使用しているリソースタイプに設定します。



リソースクォータの使用時に、オブジェクトがサーバストレージにある場合、そのオブジェクトはクォータに基づいてチャージされます。以下のクォータのタイプはストレージリソースが使い切られることから保護するのに役立ちます。

## 手順

リソースのオブジェクトカウントクォータを設定するには、以下を実行します。

1. 以下のコマンドを実行します。

```
$ oc create quota <name> \
  --hard=count/<resource>.<group>=<quota>,count/<resource>.<group>=<quota> ❶
```

- ❶ **<resource>** はリソースの名前であり、**<group>** は API グループです (該当する場合)。リソースおよびそれらの関連付けられた API グループの一覧に **oc api-resources** コマンドを使用します。

以下は例になります。

```
$ oc create quota test \
  --
  hard=count/deployments.extensions=2,count/replicasets.extensions=4,count/pods=3,count/secrets=4
resourcequota "test" created
```

この例では、一覧表示されたリソースをクラスター内の各プロジェクトのハード制限に制限します。

2. クォータが作成されていることを確認します。

```
$ oc describe quota test
Name:          test
Namespace:     quota
Resource       Used Hard
-----
count/deployments.extensions 0 2
count/pods          0 3
count/replicasets.extensions 0 4
count/secrets       0 4
```

### 5.1.6.2. 拡張リソースのリソースクォータの設定

リソースのオーバーコミットは拡張リソースには許可されません。そのため、クォータで同じ拡張リソースについて **requests** および **limits** を指定する必要があります。現時点で、プレフィックス **requests.** のあるクォータ項目のみが拡張リソースに許可されます。以下は、GPU リソース **nvdi.com/gpu** のリソースクォータを設定する方法についてのシナリオ例です。

## 手順

1. クラスター内のノードで利用可能な GPU の数を判別します。以下は例になります。

```
# oc describe node ip-172-31-27-209.us-west-2.compute.internal | egrep
'Capacity|Allocatable|gpu'
openshift.com/gpu-accelerator=true
```

```
Capacity:
  nvidia.com/gpu: 2
Allocatable:
  nvidia.com/gpu: 2
  nvidia.com/gpu 0      0
```

この例では、2つの GPU が利用可能です。

- namespace **nvidia** にクォータを設定します。この例では、クォータは **1** です。

```
# cat gpu-quota.yaml
apiVersion: v1
kind: ResourceQuota
metadata:
  name: gpu-quota
  namespace: nvidia
spec:
  hard:
    requests.nvidia.com/gpu: 1
```

- クォータを作成します。

```
# oc create -f gpu-quota.yaml
resourcequota/gpu-quota created
```

- namespace に正しいクォータが設定されていることを確認します。

```
# oc describe quota gpu-quota -n nvidia
Name:          gpu-quota
Namespace:     nvidia
Resource      Used Hard
-----
requests.nvidia.com/gpu 0   1
```

- 単一 GPU を要求する Pod を実行します。

```
# oc create -f gpu-pod.yaml

apiVersion: v1
kind: Pod
metadata:
  generateName: gpu-pod-
  namespace: nvidia
spec:
  restartPolicy: OnFailure
  containers:
  - name: rhel7-gpu-pod
    image: rhel7
    env:
    - name: NVIDIA_VISIBLE_DEVICES
      value: all
    - name: NVIDIA_DRIVER_CAPABILITIES
      value: "compute,utility"
    - name: NVIDIA_REQUIRE_CUDA
```

```

    value: "cuda>=5.0"
  command: ["sleep"]
  args: ["infinity"]
  resources:
    limits:
      nvidia.com/gpu: 1

```

6. Pod が実行されていることを確認します。

```

# oc get pods
NAME          READY   STATUS    RESTARTS   AGE
gpu-pod-s46h7 1/1     Running   0           1m

```

7. クォータ **Used** のカウンターが正しいことを確認します。

```

# oc describe quota gpu-quota -n nvidia
Name:          gpu-quota
Namespace:     nvidia
Resource       Used Hard
-----
requests.nvidia.com/gpu 1   1

```

8. **nvidia** namespace で 2 番目の GPU Pod の作成を試行します。2 つの GPU があるので、これをノード上で実行することは可能です。

```

# oc create -f gpu-pod.yaml
Error from server (Forbidden): error when creating "gpu-pod.yaml": pods "gpu-pod-f7z2w" is forbidden: exceeded quota: gpu-quota, requested: requests.nvidia.com/gpu=1, used: requests.nvidia.com/gpu=1, limited: requests.nvidia.com/gpu=1

```

クォータが 1 GPU であり、この Pod がそのクォータを超える 2 つ目の GPU の割り当てを試行したため、**Forbidden** エラーメッセージが表示されることが予想されます。

### 5.1.7. クォータの表示

Web コンソールでプロジェクトの **Quota** ページに移動し、プロジェクトのクォータで定義されるハード制限に関連する使用状況の統計を表示できます。

CLI を使用してクォータの詳細を表示することもできます。

#### 手順

1. プロジェクトで定義されるクォータの一覧を取得します。たとえば、**demoproject** というプロジェクトの場合、以下を実行します。

```

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

```

2. 関連するクォータについて記述します。たとえば、**core-object-counts** クォータの場合、以下を実行します。

```
$ oc describe quota core-object-counts -n demoproject
Name: core-object-counts
Namespace: demoproject
Resource Used Hard
----- ---- ----
configmaps 3 10
persistentvolumeclaims 0 4
replicationcontrollers 3 20
secrets 9 10
services 2 10
```

### 5.1.8. リソース消費における明示的なクォータの要求

リソースがクォータで管理されていない場合、ユーザーには消費できるリソース量の制限がありません。たとえば、gold ストレージクラスに関連するストレージのクォータがない場合、プロジェクトが作成できる gold ストレージの容量はバインドされません。

高コストのコンピュートまたはストレージリソースの場合、管理者はリソースを消費するための明示的なクォータの付与が必要となるようにする場合があります。たとえば、プロジェクトに gold ストレージクラスに関連するストレージのクォータが明示的に付与されていない場合、そのプロジェクトのユーザーはこのタイプのストレージを作成することができません。

#### 手順

リソース消費における明示的なクォータを要求するには、以下を実行します。

1. マスター設定に以下のスタanzasを追加します。

```
admissionConfig:
  pluginConfig:
    ResourceQuota:
      configuration:
        apiVersion: resourcequota.admission.k8s.io/v1alpha1
        kind: Configuration
        limitedResources:
          - resource: persistentvolumeclaims ①
        matchContains:
          - gold.storageclass.storage.k8s.io/requests.storage ②
```

- ① デフォルトで消費が制限されるグループ/リソースです。
- ② デフォルトで制限対象となる、グループ/リソースに関連付けられたクォータで追跡されるリソースの名前です。

上記の例では、クォータシステムは **PersistentVolumeClaim** を作成するか、または更新するすべての操作をインターセプトします。これは、クォータによって認識されるリソースが消費されることを確認し、プロジェクトのそれらのリソースのクォータがない場合に要求は拒否されます。

この例では、ユーザーが gold ストレージクラスに関連付けられたストレージを使用する **PersistentVolumeClaim** を作成し、プロジェクトに一致するクォータがない場合には要求は拒否されます。

### 5.1.9. クォータの同期期間の設定

リソースのセットを削除する場合、クォータの使用が回復する前にユーザーがリソースの再利用を試行すると、問題が発生する可能性があります。リソースの同期タイムフレームは、クラスター管理者が設定できる **resource-quota-sync-period** 設定によって決定されます。

再生成時間の調整は、リソースの作成および自動化が使用される場合のリソース使用状況の判別に役立ちます。



### 注記

**resource-quota-sync-period** 設定は、システムパフォーマンスのバランスを取るよう設計されています。同期期間を短縮すると、マスターに大きな負荷がかかる可能性があります。

## 手順

クォータの同期期間を設定するには、以下を実行します。

1. Kubernetes コントローラマネージャーを編集します。

```
$ oc edit kubecontrollermanager cluster
```

2. **unsupportedconfigOverrides** フィールドを、以下の設定で、**resource-quota-sync-period** フィールドに時間 (秒単位) を指定するように変更します。

```
unsupportedConfigOverrides:
  extendedArguments:
    resource-quota-sync-period:
      - 60s
```

## 5.2. 複数のプロジェクト間のリソースクォータ

ClusterResourceQuota オブジェクトで定義される複数プロジェクトのクォータは、複数プロジェクト間でクォータを共有できるようにします。それぞれの選択されたプロジェクトで使用されるリソースは集計され、その集計は選択したすべてのプロジェクトでリソースを制限するために使用されます。

以下では、クラスター管理者が複数のプロジェクトでリソースクォータを設定および管理する方法について説明します。

### 5.2.1. クォータ作成時の複数プロジェクトの選択

クォータの作成時に、アノテーションの選択、ラベルの選択、またはその両方に基づいて複数のプロジェクトを選択することができます。

## 手順

1. アノテーションに基づいてプロジェクトを選択するには、以下のコマンドを実行します。

```
$ oc create clusterquota for-user \
  --project-annotation-selector openshift.io/requester=<user_name> \
  --hard pods=10 \
  --hard secrets=20
```

これにより、以下の ClusterResourceQuota オブジェクトが作成されます。

-

```

apiVersion: v1
kind: ClusterResourceQuota
metadata:
  name: for-user
spec:
  quota: ①
  hard:
    pods: "10"
    secrets: "20"
  selector:
    annotations: ②
    openshift.io/requester: <user_name>
    labels: null ③
status:
  namespaces: ④
  - namespace: ns-one
    status:
      hard:
        pods: "10"
        secrets: "20"
      used:
        pods: "1"
        secrets: "9"
  total: ⑤
  hard:
    pods: "10"
    secrets: "20"
  used:
    pods: "1"
    secrets: "9"

```

- ① 選択されたプロジェクトに対して実施される **ResourceQuotaSpec** オブジェクトです。
- ② アノテーションの単純なキー/値のセレクターです。
- ③ プロジェクトを選択するために使用できるラベルセレクターです。
- ④ 選択された各プロジェクトの現在のクォータの使用状況を記述する namespace ごとのマップです。
- ⑤ 選択されたすべてのプロジェクトにおける使用量の総計です。

この複数プロジェクトのクォータの記述は、デフォルトのプロジェクト要求エンドポイントを使用して **<user\_name>** によって要求されるすべてのプロジェクトを制御します。ここでは、10 Pod および 20 シークレットに制限されます。

2. 同様にラベルに基づいてプロジェクトを選択するには、以下のコマンドを実行します。

```

$ oc create clusterresourcequota for-name \ ①
  --project-label-selector=name=frontend \ ②
  --hard=pods=10 --hard=secrets=20

```

- ① **clusterresourcequota** および **clusterquota** は同じコマンドのエイリアスです。 **for-name** は ClusterResourceQuota オブジェクトの名前です。

- 2 ラベル別にプロジェクトを選択するには、`--project-label-selector=key=value` 形式を使用してキーと値のペアを指定します。

これにより、以下の ClusterResourceQuota オブジェクト定義が作成されます。

```
apiVersion: v1
kind: ClusterResourceQuota
metadata:
  creationTimestamp: null
  name: for-name
spec:
  quota:
    hard:
      pods: "10"
      secrets: "20"
  selector:
    annotations: null
    labels:
      matchLabels:
        name: frontend
```

## 5.2.2. 該当する ClusterResourceQuota の表示

プロジェクト管理者は、各自のプロジェクトを制限する複数プロジェクトのクォータを作成したり、変更したりすることはできませんが、それぞれのプロジェクトに適用される複数プロジェクトのクォータを表示することはできます。プロジェクト管理者は、**AppliedClusterResourceQuota** リソースを使ってこれを実行できます。

### 手順

1. プロジェクトに適用されているクォータを表示するには、以下を実行します。

```
$ oc describe AppliedClusterResourceQuota
```

以下は例になります。

```
Name: for-user
Namespace: <none>
Created: 19 hours ago
Labels: <none>
Annotations: <none>
Label Selector: <null>
AnnotationSelector: map[openshift.io/requester:<user-name>]
Resource Used Hard
----- ----
pods      1   10
secrets   9   20
```

## 5.2.3. 選択における粒度

クォータの割り当てを要求する際にロックに関して考慮する必要があるため、複数プロジェクトのクォータで選択されるアクティブなプロジェクトの数は重要な考慮点になります。単一の複数プロジェクトクォータで100を超えるプロジェクトを選択すると、それらのプロジェクトのAPIサーバーの応答

に負の影響が及ぶ可能性があります。



## 第6章 アプリケーションの正常性のモニタリング

ソフトウェアのシステムでは、コンポーネントは一時的な問題（一時的に接続が失われるなど）、設定エラー、または外部の依存関係に関する問題などにより正常でなくなることがあります。OpenShift Container Platform アプリケーションには、正常でないコンテナを検出し、これに対応するための数多くのオプションがあります。

### 6.1. ヘルスチェックについて

プローブは実行中のコンテナで定期的に実行する Kubernetes の動作です。現時点では、2つのタイプのプローブがあり、それぞれが目的別に使用されています。

#### readiness プローブ

readiness チェックは、スケジュールの対象になるコンテナでサービス要求に対応する準備が整っているかどうかを判別します。readiness プローブがコンテナで失敗する場合、エンドポイントコントローラーはコンテナの IP アドレスがすべてのエンドポイントから削除されるようにします。readiness プローブを使用すると、コンテナが実行されていても、それがプロキシからトラフィックを受信しないようエンドポイントコントローラーに信号を送ることができます。

たとえば、readiness チェックでは、どの Pod を使用するかを制御することができます。Pod の準備ができない場合、削除されます。

#### liveness プローブ

liveness チェックは、スケジュールされているコンテナがまだ実行中であるかどうかを判断します。デッドロックなどの状態のために liveness プローブが失敗する場合、kubelet はコンテナを強制終了します。その後、コンテナは再起動ポリシーに基づいて応答します。

たとえば、**restartPolicy** として **Always** または **OnFailure** が設定されているノードでの liveness プローブは、ノード上のコンテナを強制終了してから、これを再起動します。

#### liveness チェックの例

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    test: liveness
  name: liveness-http
spec:
  containers:
  - name: liveness-http
    image: k8s.gcr.io/liveness 1
    args:
    - /server
    livenessProbe: 2
      httpGet: 3
        # host: my-host
        # scheme: HTTPS
        path: /healthz
        port: 8080
        httpHeaders:
        - name: X-Custom-Header
          value: Awesome
```

```
initialDelaySeconds: 15 ④
timeoutSeconds: 1 ⑤
name: liveness ⑥
```

- ① liveness プロブに使用するイメージを指定します。
- ② ヘルスチェックのタイプを指定します。
- ③ liveness チェックのタイプを指定します。
  - HTTP チェック。 **httpGet** を指定します。
  - コンテナー実行チェック。 **exec** を指定します。
  - TCP ソケットチェック。 **tcpSocket** を指定します。
- ④ コンテナーが起動してから最初のプロブが実行されるまでの秒数を指定します。
- ⑤ プロブ間の秒数を指定します。

### 正常ではないコンテナーについての liveness チェック出力の例

```
$ oc describe pod pod1
....

FirstSeen LastSeen  Count  From              SubobjectPath  Type    Reason  Message
-----
37s      37s      1  {default-scheduler}           Normal    Scheduled  Successfully assigned
liveness-exec to worker0
36s      36s      1  {kubelet worker0}  spec.containers{liveness}  Normal    Pulling    pulling image
"k8s.gcr.io/busybox"
36s      36s      1  {kubelet worker0}  spec.containers{liveness}  Normal    Pulled     Successfully
pulled image "k8s.gcr.io/busybox"
36s      36s      1  {kubelet worker0}  spec.containers{liveness}  Normal    Created    Created
container with docker id 86849c15382e; Security:[seccomp=unconfined]
36s      36s      1  {kubelet worker0}  spec.containers{liveness}  Normal    Started    Started
container with docker id 86849c15382e
2s       2s       1  {kubelet worker0}  spec.containers{liveness}  Warning   Unhealthy  Liveness
probe failed: cat: can't open '/tmp/healthy': No such file or directory
```

#### 6.1.1. ヘルスチェックのタイプについて

liveness チェックと readiness チェックは 3 つの方法で設定できます。

##### HTTP チェック

kubelet は web hook を使用してコンテナーの正常性を判別します。このチェックは HTTP の応答コードが 200 から 399 までの値の場合に正常とみなされます。

HTTP チェックは、これが完全に初期化されている場合は HTTP ステータスコードを返すアプリケーションに適しています。

##### コンテナー実行チェック

kubeletは、コンテナ内でコマンドを実行します。ステータス0でチェックを終了すると、成功とみなされます。

### TCP ソケットチェック

kubelet はコンテナに対してソケットを開くことを試行します。コンテナはチェックで接続を確立できる場合にのみ正常であるとみなされます。TCP ソケットチェック は、初期化が完了するまでリスニングを開始しないアプリケーションに適しています。

## 6.2. ヘルスチェックの設定

ヘルスチェックを設定するには、必要とされるチェックの種類ごとに Pod を作成します。

### 手順

ヘルスチェックを作成するには、以下の手順を実行します。

1. liveness コンテナ実行チェックを作成します。
  - a. 以下のようなYAML ファイルを作成します。

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    test: liveness
    name: liveness-exec
spec:
  containers:
  - args:
    image: k8s.gcr.io/liveness
    livenessProbe:
      exec: ❶
        command: ❷
        - cat
        - /tmp/health
      initialDelaySeconds: 15 ❸
    ...
```

- ❶ liveness チェックと liveness チェックのタイプを指定します。
- ❷ コンテナ内で使用するコマンドを指定します。
- ❸ コンテナが起動してから最初のプローブが実行されるまでの秒数を指定します。

- b. ヘルスチェック Pod の状態を確認します。

```
$ oc describe pod liveness-exec

Events:
  Type Reason Age From Message
  ---
  Normal Scheduled 9s default-scheduler Successfully assigned
  openshift-logging/liveness-exec to ip-10-0-143-40.ec2.internal
  Normal Pulling 2s kubelet, ip-10-0-143-40.ec2.internal pulling image
  "k8s.gcr.io/liveness"
```

```
Normal Pulled    1s  kubelet, ip-10-0-143-40.ec2.internal  Successfully pulled image
"k8s.gcr.io/liveness"
Normal Created  1s  kubelet, ip-10-0-143-40.ec2.internal  Created container
Normal Started  1s  kubelet, ip-10-0-143-40.ec2.internal  Started container
```

## 注記

**timeoutSeconds** パラメーターは、コンテナ実行チェックの readiness および liveness プロブには影響を与えません。OpenShift Container Platform はコンテナへの実行呼び出しでタイムアウトにならないため、タイムアウトをプロブ自体に実装できます。プロブでタイムアウトを実装する1つの方法として、**timeout** パラメーターを使用して liveness プロブおよび readiness プロブを実行できます。

```
spec:
  containers:
    livenessProbe:
      exec:
        command:
          - /bin/bash
          - '-c'
          - timeout 60 /opt/eap/bin/livenessProbe.sh ①
      timeoutSeconds: 1
      periodSeconds: 10
      successThreshold: 1
      failureThreshold: 3
```

① タイムアウト値およびプロブスクリプトへのパスです。

c. チェックを作成します。

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

2. liveness TCP ソケットチェックを作成します。

a. 以下のような YAML ファイルを作成します。

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    test: liveness
  name: liveness-tcp
spec:
  containers:
    - name: contai1 ①
      image: k8s.gcr.io/liveness
      ports:
        - containerPort: 8080 ②
      livenessProbe: ③
        tcpSocket:
```

```
port: 8080
initialDelaySeconds: 15 ④
timeoutSeconds: 1 ⑤
```

- ① ② チェックの接続先としてのコンテナの名前とポートを指定します。
- ③ liveness ヘルスチェックと liveness チェックのタイプを指定します。
- ④ コンテナが起動してから最初のプローブが実行されるまでの秒数を指定します。
- ⑤ プローブ間の秒数を指定します。

b. チェックを作成します。

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

3. readiness HTTP チェックを作成します。

a. 以下のような YAML ファイルを作成します。

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    test: readiness
    name: readiness-http
spec:
  containers:
  - args:
    image: k8s.gcr.io/readiness ①
    readinessProbe: ②
    httpGet:
      # host: my-host ③
      # scheme: HTTPS ④
      path: /healthz
      port: 8080
    initialDelaySeconds: 15 ⑤
    timeoutSeconds: 1 ⑥
```

- ① liveness プローブに使用するイメージを指定します。
- ② readiness ヘルスチェックと readiness チェックのタイプを指定します。
- ③ ホストの IP アドレスを指定します。 **host** が定義されていない場合は、 **PodIP** が使用されます。
- ④ **HTTP** または **HTTPS** を指定します。 **scheme** が定義されていない場合は、 **HTTP** スキームが使用されます。
- ⑤ コンテナが起動してから最初のプローブが実行されるまでの秒数を指定します。
- ⑥ プローブ間の秒数を指定します。

b. チェックを作成します。

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

## 第7章 アプリケーションのアイドリング

クラスター管理者は、アプリケーションをアイドリング状態にしてリソース消費を減らすことができます。これは、コストがリソース消費と関連付けられるパブリッククラウドにデプロイされている場合に役立ちます。

スケラブルなリソースが使用されていない場合、OpenShift Container Platform はリソースを検出した後にそれらを 0 レプリカに設定してアイドリングします。ネットワークトラフィックがリソースに送信される場合、レプリカをスケールアップしてアイドリング解除を実行し、通常の操作を続行します。

アプリケーションは複数のサービスや DeploymentConfig などの他のスケラブルなリソースで構成されています。アプリケーションのアイドリングには、関連するすべてのリソースのアイドリングを実行することが関係します。

### 7.1. アプリケーションのアイドリング

アプリケーションのアイドリングには、サービスに関連付けられたスケラブルなリソース (デプロイメント設定、レプリケーションコントローラーなど) を検索することが必要です。アプリケーションのアイドリングには、サービスを検索してこれをアイドリング状態としてマークし、リソースを zero レプリカにスケールダウンすることが関係します。

**oc idle** コマンドを使用して単一サービスをアイドリングするか、または **--resource-names-file** オプションを使用して複数のサービスをアイドリングすることができます。

#### 7.1.1. 単一サービスのアイドリング

##### 手順

1. 単一のサービスをアイドリングするには、以下を実行します。

```
$ oc idle <service>
```

#### 7.1.2. 複数サービスのアイドリング

複数サービスのアイドリングは、アプリケーションがプロジェクト内の一連のサービスにまたがる場合や、同じプロジェクト内で複数のアプリケーションを一括してアイドリングするため、複数サービスをスクリプトを併用してアイドリングする場合に役立ちます。

##### 手順

1. 複数サービスの一覧を含むファイルを作成します (それぞれを各行に指定)。
2. **--resource-names-file** オプションを使用してサービスをアイドリングします。

```
$ oc idle --resource-names-file <filename>
```



##### 注記

**idle** コマンドは単一プロジェクトに制限されます。クラスター全体でアプリケーションをアイドリングするには、各プロジェクトに対して **idle** コマンドを個別に実行します。

### 7.2. アプリケーションのアイドリング解除

アプリケーションサービスは、ネットワークトラフィックを受信し、直前の状態に再びスケールアップすると再びアクティブになります。これには、サービスへのトラフィックとルートを通るトラフィックの両方が含まれます。

また、アプリケーションはリソースをスケールアップすることにより、手動でアイドルリング解除することができます。

## 手順

1. DeploymentConfig をスケールアップするには、以下を実行します。

```
$ oc scale --replicas=1 dc <dc_name>
```



### 注記

現時点で、ルーターによる自動アイドルリング解除はデフォルトの HAProxy ルーターのみでサポートされています。



## 第8章 リソースを回収するためのオブジェクトのプルーニング

時間の経過と共に、OpenShift Container Platform で作成される API オブジェクトは、アプリケーションのビルドおよびデプロイなどの通常のユーザーの操作によってクラスターの etcd データストアに蓄積されます。

クラスター管理者は、不要になった古いバージョンのオブジェクトをクラスターから定期的にプルーニングできます。たとえば、イメージのプルーニングにより、使用されなくなったものの、ディスク領域を使用している古いイメージや層を削除できます。

### 8.1. プルーニングの基本操作

CLI は、共通の親コマンドでプルーニング操作を分類します。

```
$ oc adm prune <object_type> <options>
```

これにより、以下が指定されます。

- **groups**、**builds**、**deployments**、または **images** などのアクションを実行するための **<object\_type>**。
- オブジェクトタイプのプルーニングの実行においてサポートされる **<options>**。

### 8.2. グループのプルーニング

グループのレコードを外部プロバイダーからプルーニングするために、管理者は以下のコマンドを実行できます。

```
$ oc adm prune groups \
  --sync-config=path/to/sync/config [<options>]
```

表8.1 グループのプルーニング用の CLI の設定オプション

| オプション                | 説明                                 |
|----------------------|------------------------------------|
| <b>--confirm</b>     | ドライランを実行する代わりにプルーニングが実行されることを示します。 |
| <b>--blacklist</b>   | グループブラックリストファイルへのパス。               |
| <b>--whitelist</b>   | グループホワイトリストファイルへのパス。               |
| <b>--sync-config</b> | 同期設定ファイルへのパス。                      |

prune コマンドが削除するグループを表示するには、以下を実行します。

```
$ oc adm prune groups --sync-config=ldap-sync-config.yaml
```

prune 操作を実行するには、以下を実行します。

```
$ oc adm prune groups --sync-config=ldap-sync-config.yaml --confirm
```

### 8.3. デプロイメントのプルーニング

使用年数やステータスによりシステムで不要となったデプロイメントをプルーニングするために、管理者は以下のコマンドを実行できます。

```
$ oc adm prune deployments [<options>]
```

表8.2 デプロイメントのプルーニング用の CLI の設定オプション

| オプション                                             | 説明                                                                                                                                                                                                                |
|---------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>--confirm</code>                            | ドライランを実行する代わりにプルーニングが実行されることを示します。                                                                                                                                                                                |
| <code>--orphans</code>                            | DeploymentConfig を持たない、ステータスが <b>Complete</b> または <b>Failed</b> で、レプリカ数がゼロのすべてのデプロイメントをプルーニングします。                                                                                                                 |
| <code>--keep-complete=&lt;N&gt;</code>            | DeploymentConfig に基づいて、ステータスが <b>Complete</b> でレプリカ数がゼロの最後の <b>N</b> デプロイメントを維持します (デフォルト: <b>5</b> )。                                                                                                            |
| <code>--keep-failed=&lt;N&gt;</code>              | DeploymentConfig に基づいて、ステータスが <b>Failed</b> でレプリカ数がゼロの最後の <b>N</b> デプロイメントを保持します (デフォルト: <b>1</b> )。                                                                                                              |
| <code>--keep-younger-than=&lt;duration&gt;</code> | 現在の時間との対比で <code>&lt;duration&gt;</code> より後の新しいオブジェクトはプルーニングしません (デフォルト: <b>60m</b> )。有効な測定単位には、ナノ秒 ( <b>ns</b> )、マイクロ秒 ( <b>us</b> )、ミリ秒 ( <b>ms</b> )、秒 ( <b>s</b> )、分 ( <b>m</b> )、および時間 ( <b>h</b> ) が含まれます。 |

プルーニング操作によって削除されるものを確認するには、以下を実行します。

```
$ oc adm prune deployments --orphans --keep-complete=5 --keep-failed=1 \
--keep-younger-than=60m
```

プルーニング操作を実際に実行するには、以下を実行します。

```
$ oc adm prune deployments --orphans --keep-complete=5 --keep-failed=1 \
--keep-younger-than=60m --confirm
```

### 8.4. ビルドのプルーニング

使用年数やステータスによりシステムで不要となったビルドをプルーニングするために、管理者は以下のコマンドを実行できます。

```
$ oc adm prune builds [<options>]
```

表8.3 ビルドのプルーニング用の CLI の設定オプション

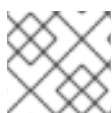
| オプション                                       | 説明                                                                                                      |
|---------------------------------------------|---------------------------------------------------------------------------------------------------------|
| <b>--confirm</b>                            | ドライランを実行する代わりにプルーニングが実行されることを示します。                                                                      |
| <b>--orphans</b>                            | ビルド設定が存在せず、ステータスが complete (完了)、failed (失敗)、error (エラー)、または canceled (中止) のすべてのビルドをプルーニングします。           |
| <b>--keep-complete=&lt;N&gt;</b>            | ビルド設定に基づいて、ステータスが complete (完了) の最後の <b>N</b> ビルドを保持します (デフォルト: <b>5</b> )。                             |
| <b>--keep-failed=&lt;N&gt;</b>              | ビルド設定に基づいて、ステータスが failed (失敗)、error (エラー)、または canceled (中止) の最後の <b>N</b> ビルドを保持します (デフォルト: <b>1</b> )。 |
| <b>--keep-younger-than=&lt;duration&gt;</b> | 現在の時間との対比で <b>&lt;duration&gt;</b> より後の新しいオブジェクトはプルーニングしません (デフォルト: <b>60m</b> )。                       |

プルーニング操作によって削除されるものを確認するには、以下を実行します。

```
$ oc adm prune builds --orphans --keep-complete=5 --keep-failed=1 \
--keep-younger-than=60m
```

プルーニング操作を実際に行うには、以下を実行します。

```
$ oc adm prune builds --orphans --keep-complete=5 --keep-failed=1 \
--keep-younger-than=60m --confirm
```



### 注記

開発者は、ビルドの設定を変更して自動ビルドプルーニングを有効にできます。

### 追加リソース

- [Performing advanced builds → Pruning builds](#)

## 8.5. イメージのプルーニング

使用年数やステータスまたは制限の超過によりシステムで不要となったイメージをプルーニングするために、管理者は以下のコマンドを実行できます。

```
$ oc adm prune images [<options>]
```

現時点でイメージをプルーニングするには、まずアクセストークンを使ってユーザーとして CLI にログインする必要があります。ユーザーにはクラスターロール **system:image-pruner** 以上のロールがなければなりません (例: **cluster-admin**)。

**--prune-registry=false** が使用されていない限り、イメージのプルーニングにより、統合レジストリーのデータが削除されます。

**--namespace** フラグの付いたイメージをプルニングしてもイメージは削除されず、イメージストリームのみが削除されます。イメージは namespace を使用しないリソースです。そのため、プルニングを特定の namespace に制限すると、イメージの現在の使用量を算出できなくなります。

デフォルトで、統合レジストリーは Blob メタデータをキャッシュしてストレージに対する要求数を減らし、要求の処理速度を高めます。プルニングによって統合レジストリーのキャッシュが更新されることはありません。プルニング後にプッシュされる、プルニングされた層を含むイメージは破損します。キャッシュにメタデータを持つプルニングされた層はプッシュされないためです。これは、レジストリーの再デプロイによって実行できます。

```
# oc patch deployment image-registry -n openshift-image-registry --type=merge --patch="{\"spec\": {\"template\": {\"metadata\": {\"annotations\": {\"kubectl.kubernetes.io/restartedAt\": \"$(date '+%Y-%m-%dT%H:%M:%SZ' -u)}\"}}}}"
```

統合レジストリーが Redis キャッシュを使用する場合、データベースを手動でクリーンアップする必要があります。

**oc adm prune images** 操作ではレジストリーのルートが必要です。レジストリーのルートはデフォルトでは作成されません。レジストリーのルートの作成方法に関する詳細は「[Image Registry Operator in OpenShift Container Platform](#)」を参照し、レジストリーサービスを公開する方法の詳細は「[Exposing the registry](#)」を参照してください。

表8.4 イメージのプルニング用の CLI の設定オプション

| オプション                                       | 説明                                                                                                                                               |
|---------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>--all</b>                                | レジストリーにプッシュされていないものの、プルスルー (pullthrough) でミラーリングされたイメージを組み込みます。これはデフォルトでオンに設定されます。プルニングを統合レジストリーにプッシュされたイメージに制限するには、 <b>--all=false</b> を渡します。   |
| <b>--certificate-authority</b>              | OpenShift Container Platform で管理されるレジストリーと通信する際に使用する認証局ファイルへのパスです。デフォルトは現行ユーザーの設定ファイルの認証局データに設定されます。これが指定されている場合、セキュアな通信が実行されます。                 |
| <b>--confirm</b>                            | ドライランを実行する代わりにプルニングが実行されることを示します。これには、統合コンテナイメージレジストリーへの有効なルートが必要になります。このコマンドがクラスターネットワーク外で実行される場合、ルートは <b>--registry-url</b> を使用して指定される必要があります。 |
| <b>--force-insecure</b>                     | このオプションは注意して使用してください。HTTP 経由でホストされるか、または無効な HTTPS 証明書を持つコンテナレジストリーへの非セキュアな接続を許可します。                                                              |
| <b>--keep-tag-revisions=&lt;N&gt;</b>       | それぞれのイメージストリームについては、タグごとに最大 <b>N</b> のイメージリビジョンを保持します (デフォルト: <b>3</b> )。                                                                        |
| <b>--keep-younger-than=&lt;duration&gt;</b> | 現在の時間との対比で <b>&lt;duration&gt;</b> 未満の新しいイメージはプルニングしません。現在の時間との対比で <b>&lt;duration&gt;</b> 未満の他のオブジェクトで参照されるイメージはプルニングしません (デフォルト: <b>60m</b> )。 |

| オプション                          | 説明                                                                                                                                                                                                                                                                                                                                  |
|--------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>--prune-over-size-limit</b> | 同じプロジェクトに定義される最小の制限を超える各イメージをプルーニングします。このフラグは <b>--keep-tag-revisions</b> または <b>--keep-younger-than</b> と共に使用することはできません。                                                                                                                                                                                                           |
| <b>--registry-url</b>          | レジストリーと通信する際に使用するアドレスです。このコマンドは、管理されるイメージおよびイメージストリームから判別されるクラスター内の URL の使用を試行します。これに失敗する (レジストリーを解決できないか、これにアクセスできない) 場合、このフラグを使用して他の機能するルートを指定する必要があります。レジストリーのホスト名の前には、特定の接続プロトコルを実施する <b>https://</b> または <b>http://</b> を付けることができます。                                                                                           |
| <b>--prune-registry</b>        | 他のオプションで規定される条件と共に、このオプションは、OpenShift Container Platform イメージ API オブジェクトに対応するレジストリーのデータがプルーニングされるかどうかを制御します。デフォルトで、イメージのプルーニングは、イメージ API オブジェクトとレジストリーの対応するデータの両方を処理します。このオプションは、イメージオブジェクトの数を減らすなどの目的で etcd の内容のみを削除することを検討して (ただしレジストリーのストレージのクリーンアップは検討していない場合)、レジストリーの適切なメンテナンス期間中などにレジストリーのハードプルーニングによってこれを別途実行しようとする場合に役立ちます。 |

### 8.5.1. イメージのプルーニングの各種条件

- **--keep-younger-than** 分前よりも後に作成され、現時点で以下によって参照されていない OpenShift Container Platform で管理されるイメージ (アノテーション **openshift.io/image.managed** を持つイメージ) を削除します。
  - **--keep-younger-than** 分前よりも後に作成された Pod
  - **--keep-younger-than** 分前よりも後に作成されたイメージストリーム
  - 実行中の Pod
  - 保留中の Pod
  - ReplicationController
  - 任意のデプロイメント
  - DeploymentConfig
  - 任意の ReplicaSet
  - ビルド設定
  - ビルド
  - **stream.status.tags[].items** の **--keep-tag-revisions** の最新のアイテム

- 同じプロジェクトで定義される最小の制限を超えており、現時点で以下によって参照されていない OpenShift Container Platform で管理されるイメージ (アノテーション **openshift.io/image.managed** を持つイメージ) を削除します。
  - 実行中の Pod
  - 保留中の Pod
  - ReplicationController
  - 任意のデプロイメント
  - DeploymentConfig
  - 任意の ReplicaSet
  - ビルド設定
  - ビルド
- 外部レジストリーからのプルニングはサポートされていません。
- イメージがプルニングされる際、イメージのすべての参照は **status.tags** にイメージの参照を持つすべてのイメージストリームから削除されます。
- イメージによって参照されなくなったイメージ層は削除されます。



### 注記

**--prune-over-size-limit** フラグは **--keep-tag-revisions** または **--keep-younger-than** フラグと共に使用することができません。これを実行すると、この操作が許可されないことを示す情報が返されます。

**--prune-registry=false** とその後にレジストリーのハードプルニングを実行することで、OpenShift Container Platform イメージ API オブジェクトの削除とイメージデータのレジストリーからの削除を分離することができます。これにより、タイミングウィンドウが制限され、1つのコマンドで両方をプルニングする場合よりも安全に実行できるようになります。ただし、タイミングウィンドウを完全に取り除くことはできません。

たとえばプルニングの実行時にプルニング対象のイメージを特定する場合も、そのイメージを参照する Pod を引き続き作成することができます。また、プルニングの操作時にイメージを参照している可能性のある API オブジェクトを追跡することもできます。これにより、削除されたコンテンツの参照に関連して発生する可能性のある問題を軽減することができます。

また、**--prune-registry** オプションを指定しないか、または **--prune-registry=true** を指定してプルニングを再実行しても、**--prune-registry=false** を指定して以前にプルニングされたイメージの、イメージレジストリー内で関連付けられたストレージがプルニングされる訳ではないことに注意してください。**--prune-registry=false** を指定してプルニングされたすべてのイメージは、レジストリーのハードプルニングによってのみ削除できます。

## 8.5.2. イメージのプルニング操作の実行

### 手順

1. プルニング操作によって削除されるものを確認するには、以下を実行します。

○ 最高3つのタグリビジョンを保持し、6分前よりも後に作成されたリソース (イメージ、イ

- d. 取回すもののタグレvisionsを保持し、0日前より後に作成されたイメージ (イメージ、イメージストリームおよび Pod) を保持します。

```
$ oc adm prune images --keep-tag-revisions=3 --keep-younger-than=60m
```

- b. 定義された制限を超えるすべてのイメージをプルーニングします。

```
$ oc adm prune images --prune-over-size-limit
```

2. 前述のステップからオプションを指定してプルーニングの操作を実際に実行するには、以下を実行します。

```
$ oc adm prune images --keep-tag-revisions=3 --keep-younger-than=60m --confirm
```

```
$ oc adm prune images --prune-over-size-limit --confirm
```

### 8.5.3. セキュアまたは非セキュアな接続の使用

セキュアな通信の使用は優先され、推奨される方法です。これは、必須の証明書検証と共に HTTPS 経由で実行されます。**prune** コマンドは、可能な場合は常にセキュアな通信の使用を試行します。これを使用できない場合には、非セキュアな通信にフォールバックすることがあり、これには危険が伴います。この場合、証明書検証は省略されるか、または単純な HTTP プロトコルが使用されます。

非セキュアな通信へのフォールバックは、**--certificate-authority** が指定されていない場合、以下のケースで可能になります。

1. **prune** コマンドが **--force-insecure** オプションと共に実行される。
2. 指定される **registry-url** の前に **http://** スキームが付けられる。
3. 指定される **registry-url** はローカルリンクアドレスまたは **localhost** である。
4. 現行ユーザーの設定が非セキュアな接続を許可する。これは、ユーザーが **--insecure-skip-tls-verify** を使用してログインするか、またはプロンプトが出される際に非セキュアな接続を選択することによって生じる可能性があります。



#### 重要

レジストリーのセキュリティが、OpenShift Container Platform で使用されるものとは異なる認証局で保護される場合、これを **--certificate-authority** フラグを使用して指定する必要があります。そうしない場合、**prune** コマンドがエラーを出して失敗します。

### 8.5.4. イメージのプルーニングに関する問題

#### イメージがプルーニングされない

イメージが蓄積し続け、**prune** コマンドが予想よりも小規模な削除を実行する場合、プルーニング候補のイメージについて満たすべきイメージプルーニングの条件があることを確認します。

とくに削除する必要のあるイメージが、それぞれのタグ履歴において選択したタグレvisionsのしきい値よりも高い位置にあることを確認します。たとえば、**sha:abz** という名前の古く陳腐化したイメージがあるとします。イメージがタグ付けされている namespace **N** で以下のコマンドを実行すると、イメージが **myapp** という単一イメージストリームで3回タグ付けされていることに気づかれるでしょう。

```
$ image_name="sha:abz"
$ oc get is -n N -o go-template='{{range $isi, $is := .items}}{{range $ti, $tag := $is.status.tags}}\
  {{range $ii, $item := $tag.items}}{{if eq $item.image ""${image_name}}\
  $"}}{{ $is.metadata.name }}:{{ $tag.tag }} at position {{ $ii }} out of {{ len $tag.items }}\n\
  {{end}}{{end}}{{end}}{{end}}'
myapp:v2 at position 4 out of 5
myapp:v2.1 at position 2 out of 2
myapp:v2.1-may-2016 at position 0 out of 1
```

デフォルトオプションが使用される場合、イメージは **myapp:v2.1-may-2016** タグの履歴の **0** の位置にあるためプルーニングされません。イメージがプルーニングの対象とみなされるようにするには、管理者は以下を実行する必要があります。

- **oc adm prune images** コマンドで **--keep-tag-revisions=0** を指定します。

### 注意

このアクションを実行すると、イメージが指定されたしきい値よりも新しいか、またはこれよりも新しいオブジェクトによって参照されていない限り、すべてのタグが基礎となるイメージと共にすべての namespace から削除されます。

- リビジョンのしきい値の下にあるすべての **istags**、つまり **myapp:v2.1** および **myapp:v2.1-may-2016** を削除します。
- 同じ **istag** にプッシュする新規ビルドを実行するか、または他のイメージをタグ付けしてイメージを履歴内でさらに移動させます。ただし、これは古いリリースタグの場合には常に適切な操作となる訳ではありません。

特定のイメージのビルド日時が名前の一部になっているタグは、その使用を避ける必要があります (イメージが未定義の期間保持される必要がある場合を除きます)。このようなタグは履歴内で1つのイメージのみに関連付けられる可能性があり、その場合にこれらをプルーニングできなくなります。

### 非セキュアなレジストリーに対するセキュアな接続の使用

**oc adm prune images** コマンドの出力で以下のようなメッセージが表示される場合、レジストリーのセキュリティは保護されておらず、**oc adm prune images** クライアントがセキュアな接続の使用を試行することを示しています。

```
error: error communicating with registry: Get https://172.30.30.30:5000/healthz: http: server gave HTTP response to HTTPS client
```

1. 推奨される解決法として、レジストリーのセキュリティを保護することができます。そうしない場合は、**--force-insecure** をコマンドに追加して、クライアントに対して非セキュアな接続の使用を強制することができますが、これは推奨される方法ではありません。

### セキュリティが保護されたレジストリーに対する非セキュアな接続の使用

**oc adm prune images** コマンドの出力に以下のエラーのいずれかが表示される場合、レジストリーのセキュリティ保護に使用されている認証局で署名された証明書が、接続の検証用に **oc adm prune images** クライアントで使用されるものとは異なることを意味します。

```
error: error communicating with registry: Get http://172.30.30.30:5000/healthz: malformed HTTP response "\x15\x03\x01\x00\x02\x02"
error: error communicating with registry: [Get https://172.30.30.30:5000/healthz: x509: certificate signed by unknown authority, Get http://172.30.30.30:5000/healthz: malformed HTTP response "\x15\x03\x01\x00\x02\x02"]
```



デフォルトでは、ユーザーの接続ファイルに保存されている認証局データが使用されます。これはマスター API との通信の場合も同様です。

**--certificate-authority** オプションを使用してコンテナイメージレジストリーサーバーに適切な認証局を指定します。

### 正しくない認証局の使用

以下のエラーは、セキュリティーが保護されたコンテナイメージレジストリーの証明書の署名に使用される認証局がクライアントで使用される認証局とは異なることを示しています。

```
error: error communicating with registry: Get https://172.30.30.30:5000/: x509: certificate signed by unknown authority
```

フラグ **--certificate-authority** を使用して適切な認証局を指定します。

回避策として、**--force-insecure** フラグを代わりに追加することもできます。ただし、これは推奨される方法ではありません。

### 追加リソース

- [レジストリーへのアクセス](#)
- [レジストリーの公開](#)

## 8.6. レジストリーのハードプルーニング

OpenShift Container レジストリーは、OpenShift Container Platform クラスターの etcd で参照されない Blob を蓄積します。基本的なイメージプルーニングの手順はこれらに対応しません。これらの Blob は **孤立した Blob** と呼ばれています。

孤立した Blob は以下のシナリオで発生する可能性があります。

- **oc delete image <sha256:image-id>** コマンドを使ってイメージを手動で削除すると、etcd のイメージのみが削除され、レジストリーのストレージからは削除されません。
- デーモンの障害によって生じるレジストリーへのプッシュにより、一部の Blob はアップロードされるものの、(最後のコンポーネントとしてアップロードされる) イメージマニフェストはアップロードされません。固有のイメージ Blob すべてが孤立します。
- OpenShift Container Platform がクォータの制限によりイメージを拒否します。
- 標準のイメージプルーナーがイメージマニフェストを削除するが、関連する Blob を削除する前に中断されます。
- 対象の Blob を削除できないというレジストリープルーナーのバグにより、それらを参照するイメージオブジェクトは削除され、Blob は孤立します。

基本的なイメージプルーニングとは異なるレジストリーの **ハードプルーニング** により、クラスター管理者は孤立した Blob を削除することができます。OpenShift Container レジストリーのストレージ領域が不足している場合や、孤立した Blob があると思われる場合にはハードプルーニングを実行する必要があります。

これは何度も行う操作ではなく、多数の孤立した Blob が新たに作成されているという証拠がある場合にのみ実行する必要があります。または、(作成されるイメージの数によって異なりますが) 1日1回などの定期的な間隔で標準のイメージプルーニングを実行することもできます。

## 手順

孤立した Blob をレジストリーからハードプルーニングするには、以下を実行します。

### 1. ログイン

CLI で **kubeadmin** として、または **openshift-image-registry** namespace へのアクセスのある別の特権ユーザーとしてクラスターにログインします。

### 2. 基本的なイメージプルーニングの実行

基本的なイメージプルーニングにより、不要になった追加のイメージが削除されます。ハードプルーニングによってイメージが削除される訳ではありません。レジストリーストレージに保存された Blob のみが削除されます。したがって、ハードプルーニングの実行前にこれを実行する必要があります。

### 3. レジストリーの読み取り専用モードへの切り替え

レジストリーが読み取り専用モードで実行されていない場合、プルーニングと同時に実行されているプッシュの結果は以下のいずれかになります。

- 失敗する。孤立した Blob が新たに発生します。
- 成功する。ただし、(参照される Blob の一部が削除されたため) イメージをプルできません。

プッシュは、レジストリーが読み取り書き込みモードに戻されるまで成功しません。したがって、ハードプルーニングは注意してスケジューリングする必要があります。

レジストリーを読み取り専用モードに切り換えるには、以下を実行します。

- a. **configs.imageregistry.operator.openshift.io/cluster** で、 **spec.readOnly** を **true** に設定します。

```
$ oc patch configs.imageregistry.operator.openshift.io/cluster -p '{"spec": {"readOnly":true}}' --type=merge
```

### 4. system:image-pruner ロールの追加

一部のリソースを一覧表示するには、レジストリーインスタンスの実行に使用するサービスアカウントに追加のパーミッションが必要になります。

- a. サービスアカウント名を取得します。

```
$ service_account=$(oc get -n openshift-image-registry \
  -o jsonpath='{.spec.template.spec.serviceAccountName}' deploy/image-registry)
```

- b. **system:image-pruner** クラスターロールをサービスアカウントに追加します。

```
$ oc adm policy add-cluster-role-to-user \
  system:image-pruner -z \
  ${service_account} -n openshift-image-registry
```

### 5. (オプション) プルーナーのドライランモードでの実行

削除される Blob の数を確認するには、ドライランモードでハードプルーナーを実行します。実際の変更は加えられません。

```
$ oc -n openshift-image-registry \
  rsh deploy/image-registry \
  /usr/bin/dockerregistry -prune=check
```

または、プルーニング候補の実際のパスを取得するには、ロギングレベルを上げます。

```
$ oc -n openshift-image-registry \
  rsh deploy/image-registry env REGISTRY_LOG_LEVEL=info \
  /usr/bin/dockerregistry -prune=check
```

### 出力サンプル (切り捨て済み)

```
$ oc exec image-registry-3-vhndw \
  -- /bin/sh -c 'REGISTRY_LOG_LEVEL=info /usr/bin/dockerregistry -prune=check'

time="2017-06-22T11:50:25.066156047Z" level=info msg="start prune (dry-run mode)"
distribution_version="v2.4.1+unknown" kubernetes_version=v1.6.1+${Format:%h$}
openshift_version=unknown
time="2017-06-22T11:50:25.092257421Z" level=info msg="Would delete blob:
sha256:00043a2a5e384f6b59ab17e2c3d3a3d0a7de01b2cabeb606243e468acc663fa5"
go.version=go1.7.5 instance.id=b097121c-a864-4e0c-ad6c-cc25f8fdf5a6
time="2017-06-22T11:50:25.092395621Z" level=info msg="Would delete blob:
sha256:0022d49612807cb348cab562c072ef34d756adfe0100a61952cbcb87ee6578a"
go.version=go1.7.5 instance.id=b097121c-a864-4e0c-ad6c-cc25f8fdf5a6
time="2017-06-22T11:50:25.092492183Z" level=info msg="Would delete blob:
sha256:0029dd4228961086707e53b881e25eba0564fa80033fbbb2e27847a28d16a37c"
go.version=go1.7.5 instance.id=b097121c-a864-4e0c-ad6c-cc25f8fdf5a6
time="2017-06-22T11:50:26.673946639Z" level=info msg="Would delete blob:
sha256:ff7664dfc213d6cc60fd5c5f5bb00a7bf4a687e18e1df12d349a1d07b2cf7663"
go.version=go1.7.5 instance.id=b097121c-a864-4e0c-ad6c-cc25f8fdf5a6
time="2017-06-22T11:50:26.674024531Z" level=info msg="Would delete blob:
sha256:ff7a933178ccd931f4b5f40f9f19a65be5eeec207e4fad2a5bafd28afbef57e"
go.version=go1.7.5 instance.id=b097121c-a864-4e0c-ad6c-cc25f8fdf5a6
time="2017-06-22T11:50:26.674675469Z" level=info msg="Would delete blob:
sha256:ff9b8956794b426cc80bb49a604a0b24a1553aae96b930c6919a6675db3d5e06"
go.version=go1.7.5 instance.id=b097121c-a864-4e0c-ad6c-cc25f8fdf5a6
...
Would delete 13374 blobs
Would free up 2.835 GiB of disk space
Use -prune=delete to actually delete the data
```

## 6. ハードプルーニングの実行

ハードプルーニングを実行するには、**image-registry** Pod の実行中のインスタンスで以下のコマンドを実行します。

```
$ oc -n openshift-image-registry \
  rsh deploy/image-registry \
  /usr/bin/dockerregistry -prune=delete
```

### 出力サンプル

```
$ oc exec image-registry-3-vhndw \
  -- /usr/bin/dockerregistry -prune=delete
```

```
Deleted 13374 blobs  
Freed up 2.835 GiB of disk space
```

#### 7. レジストリーを読み取り/書き込みモードに戻す

プルーニングの終了後は、レジストリーを読み取り/書き込みモードに戻すことができます。 `configs.imageregistry.operator.openshift.io/cluster` で、 `spec.readOnly` を `false` に設定します。

```
$ oc patch configs.imageregistry.operator.openshift.io/cluster -p '{"spec":{"readOnly":false}}' -  
-type=merge
```

## 8.7. CRON ジョブのプルーニング

cron ジョブは正常なジョブのプルーニングを実行できますが、失敗したジョブを適切に処理していない可能性があります。そのため、クラスター管理者はジョブの定期的なクリーンアップを手動で実行する必要があります。また、信頼できるユーザーの小規模なグループに cron ジョブへのアクセスを制限し、cron ジョブでジョブや Pod が作成され過ぎないように適切なクォータを設定する必要もあります。

### 追加リソース

- [ジョブを使用した Pod でのタスクの実行](#)
- [複数のプロジェクト間のリソースクォータ](#)
- [RBAC の使用によるパーミッションの定義および適用](#)