Red Hat Advanced Cluster Management for Kubernetes 2.0

Manage applications

Manage applications

Last Updated: 2021-06-10

Red Hat Advanced Cluster Management for Kubernetes 2.0 Manage applications

Manage applications

Legal Notice

Abstract

Manage applications in Red Hat Advanced Cluster Management for Kubernetes

Table of Contents

CHAPTER 1. MANAGING APPLICATIONS

Review the following topics to learn more about creating, deploying, and managing your applications. This guide assumes familiarity with Kubernetes concepts and terminology. Key Kubernetes terms and components are not defined. For more information about Kubernetes concepts, see Kubernetes Documentation.

The application management functions provide you with unified and simplified options for constructing and deploying applications and application updates. With these functions, your developers and DevOps personnel can create and manage applications across environments through channel and subscription-based automation.

See the following topics:

- Application management lifecycle
- Application model and definitions
- Application resources
- Managing applications with the console
- Creating and managing channels
- Creating and managing subscriptions
- Creating and managing placement rules
- Creating and managing application resources
- Deploying by using application resources
- Application resource samples

1.1. APPLICATION MANAGEMENT LIFECYCLE

The application model is based on subscribing to one or more Kubernetes resource repositories (*channel* resource) that contains resources that are deployed on managed clusters.

The *subscription* component uses a *placement rule* resource to define the managed clusters where the Kubernetes resource will be deployed.

The last piece in the application model is the *application* resource that references one or more repositories subscriptions. The purpose for the application is to group deployed Kubernetes resources and provide data aggregation that is easy to understand and manage.

Learn more about the Application lifecycle from the following topics:

- Application model and definitions
- Managing applications with the console

1.1.1. Application model and definitions

The Application model consists of the following Kubernetes custom resources: channels, subscription, placement rules, and applications.

- Channels (**channel.apps.open-cluster-management.io**) define the source repositories that a cluster can subscribe to with a subscription and can be the following types: Git repositories, Helm release registries, objectstores, and resource template (deployable) namespace on the hub cluster.
  Note: It is best practice to create each channel in a uique namespace. However, a Git channel can share a namespace with another type of channel, including Git, Helm, Kubernetes Namespace, and Object store.

- Subscriptions (**subscription.apps.open-cluster-management.io**) allow clusters to subscribe to a source repository (channel) that can be the following types: Git repository, Helm release registry, objectstore, or resource template (deployable) namespace. Subscriptions can be applied locally to the hub or to managed-clusters.

- Placement rules (**placementrule.apps.open-cluster-management.io**) define the target clusters where subscriptions deploy and maintain the Kubernetes resources. You can use placement rules to help you facilitate the multi-cluster deployment. Placement rules can be shared across subscriptions.

- Applications (**application.app.k8s.io**) in Red Hat Advanced Cluster Management for Kubernetes are used for grouping Kubernetes resources that make up an application.

For more information about creating and managing application resources, see:

- Creating and managing channels
- Creating and managing subscriptions
- Creating and managing placement rules
- Creating and managing application resources

1.1.2. Managing applications with the console

The console includes a dashboard for managing the application lifecycle. You can use the console dashboard to create and manage applications resources. You can also view the status of your applications. The dashboard includes enhanced capabilities, which your developers and operations personnel can use to create, deploy, update, manage, and visualize applications across your clusters.

See the following application console capabilities:

- Use the topology view to visualize deployed applications across your clusters, including any associated channels and subscriptions.

- Access a topology view that encompasses the new application resource definitions, including channels, subscriptions, and placement rules.

- View individual status in the context of an application, including deployments, updates, and subscriptions.

- Add and edit channels, subscriptions, placement rules, and applications.

Samples for all resources are located in the Application resource samples documentation.

The console includes different tools that each provide different application management capabilities. These capabilities allow you to easily create, find, update, and deploy applications resources.

- Applications dashboards

- Search
- Resource topology

1.1.2.1. Applications dashboards

From the main *Applications* dashboard, you can view information about all applications, and you can select and view information about a specific application.

From the *Overview* tab on this dashboard, you can complete the following tasks for all applications:

- View a table that lists all applications.
- Use the *Find resources* box to filter the applications that are listed.
- View the application name and namespace.
- View the number of managed clusters where the application is deployed through a subscription.
- View number of subscriptions on the *hub* cluster that are used to deploy the application and the associated status.
- View the date when the application was created.
- Click **Options** for more actions, such as **Delete application**.
- Click on an application name in the table to view more details about that specific resource.
- Access the resource pipeline from the *Resources* tab.
- To see *Resources*, click on **Subscription** to go to the *Search* page, then see hub cluster subscriptions and all subscriptions that are deployed to managed clusters.
- View the summary of resources for all applications on different cards. These resource summary cards show the following information about your applications and related managed clusters:
    - The number of subscriptions on the hub cluster and the number of subscriptions that failed to deploy to the hub cluster.
    - The total number of managed clusters.
    - The number of subscriptions that are active for those managed clusters.
    - The total number of channels and placement rules for all applications.

You can click each summary card to open the *Search* page and display more information about the selected resource.

- From this page, you can also click the multiple **Create** options to create more resources with YAML editors.
  When you select to create any of these resources a YAML editor opens for you to use to define the resource. Default sample YAML content is included as templates to indicate the required fields to help you create any of these resource types.

  View resources for all applications, which includes a table that displays details for your applications and other resources:

- Each row provides details for a single application. Rows are included for all of your applications. If needed, you can use the search box to filter the application rows to find a matching application.
- Expand the row for each application to view more details for the associated channels, subscriptions, and placement rules, view related incidents, and view the status of resource (pod) deployments.
- Select an associated subscription for a channel and application to view more details about that subscription, including the subscribed deployables, the associated channel, namespace, and host-cluster, the status of any related resource (pod) deployments, and the placement settings that are used with the subscription.

1.1.2.1.1. Application dashboard (single applications)

From the *Overview* tab of the dashboard for a single application, you can complete the following tasks:

- View the summary of resource highlights for the application on different cards. These resource summary cards show the following information:
  - The number of subscriptions on the hub cluster for the application and the number of subscriptions that failed to deploy to the hub cluster.
  - The total number of managed clusters where the application is used. This card also shows the total number of subscriptions for the application on those managed clusters, and the number of subscriptions that failed to deploy to those managed clusters.
  - The number of pods for the application, including the number that are running and failed.
  - The number of policy violations and incidents that are related to the application. You can click each summary card to open the *Search* page and display more information about the selected resource.
- Select to view additional details for the application, including the application namespace, version, creation date, associated labels and annotations, and more.
- View the resource topology for the application that shows the application and related components such as channels, deployables, related services, pods, and placement rules. You can select to edit the topology and update the YAML definitions for the application and related resources, such as to add, change, or remove resources.

From the *Resources* tab for a single application, you can complete the following tasks:

- View the resource list for the application. This list includes all resources that are associated with the application, including pods, secrets, services, and more. This list indicates the name, namespace, kind, API groups, and status of each resource. The list also indicates the cluster where the resource is deployed, the creation date of the resource, and the last date that the resource was updated.
- View the resource pipeline for the application, which includes the summary of the application resources, options to create resources, and a table that provides more details about the resources. The resource pipeline for a single application includes similar information to the pipeline for all applications. The difference between the two pipelines is that this pipeline is scoped to just the selected application.

1.1.2.2. Search

The console *Search* page supports searching for application resources by the component **kind** for each resource. To search for resources, use the following values:

Application resource	Kind (search parameter)
Application	**Application**
Channel	**Channel**
Deployable	**Deployable**
Secret	**Secret**
Placement rule	**PlacementRule**
Subscription	**Subscription**

You can also search by other fields, including name, namespace, cluster, label, and more.

From the search results, you can view identifying details for each resource, including the name, namespace, cluster, labels, and creation date.

If needed, you can also expand the *Options* menu in the search results for a resource to select to delete that resource.

By clicking the resource name in the search results, a YAML editor opens and displays the YAML definition for the resource. You can choose to edit the definition within the editor. Any changes that you save are applied to the resource immediately.

For more information about using search, see Search in the console.

1.1.2.3. Resource topology

The application topology includes a visualization of application cards that display status. The topology view for each application includes any services, deployments, charts, and pods for that application.

- You can select any component from the topology view to view more details.
- You can hover your cursor over a resource to view the component kind, name, and namespace and links to view the search results for the resource or namespace.
- View the details for a pod. You can select to view the logs for that pod.
- View cluster CPU and memory.
  Note: The cluster CPU and memory percentage that is displayed is the percentage that is currently utilized. This value is rounded down, so a very small value might display as **0**.

  = Application resources

Within Red Hat Advanced Cluster Management for Kubernetes, applications are composed of multiple application resources. The foundational resources for Red Hat Advanced Cluster Management for Kubernetes applications are the **application** resource and the **deployable** resource.

In addition, you can use channel, subscription, and placement rule resources to help you deploy, update, and manage your overall applications.

Both single and multi-cluster applications use the same Kubernetes specifications, but multi-cluster applications involve more automation of the deployment and application management lifecycle.

All of the application component resources for Red Hat Advanced Cluster Management for Kubernetes applications are defined in YAML file spec sections. When you need to create or update an application component resource, you need to create or edit the appropriate spec section to include the labels for defining your resource.

If you have applications that require secret resources to operate, you can use subscriptions to deploy the secrets to the managed clusters where the resources are needed.

If you have applications that require Kubernetes resources or Helm charts from channels that require authorization, such as entitled Git repositories, you can use secrets to provide access to these channels. You can include a reference to a secret within your subscriptions to provide your subscriptions the required credentials to access secure channels. With this access, your subscriptions can access Kubernetes resources and Helm charts for deployment from these channels while maintaining data security.

View the following application resource sections:

1.1.3. Channels

Channels (**channel.apps.open-cluster-management.io**) provide you with improved continuous integration and continuous delivery capabilities for creating and managing your Red Hat Advanced Cluster Management for Kubernetes applications. Channels are custom resource definitions that can help you streamline deployments and separate cluster access.

For more information about creating and managing channels, see Creating and managing channels. See Application resource samples for sample YAML files.

Channels define a namespace within the hub cluster and point to a physical place where resources are stored for deployment, such as an object store, Kubernetes namespace, Helm repository, or Git repository. Clusters can subscribe to channels for identifying the deployables to deploy to each cluster.

- Deployables within a channel can be accessed by only the clusters that subscribe to that channel.
  The placement, overrides, and dependencies for deployables are defined within the spec for the deployables. The placement for deployables can also be defined within a subscription, such as for multi-cluster deployments.

- Secrets are Kubernetes resources that you can use to store authorization and other sensitive information, such as passwords, OAuth tokens, and SSH keys.
  By storing this information as secrets, you can separate the information from the application components that require the information to improve your data security.

See Managing secrets. See Secret samples for sample YAML files.

For **Namespace** and **ObjectBucket** channel types, the spec for each channel can define conditions that a deployable must match to be included in the channel. These conditions are defined as Kubernetes labels for the channel, such as the source namespace, package name, labels, and annotations. A deployable must have the same labels for the deployable to be included in the channel. A deployable can be included in a channel only when the deployable is labeled with the same labels as the channel.

1.1.4. Subscriptions

As with channels, subscriptions (**subscription.apps.open-cluster-management.io**) provide you with improved continuous integration and continuous delivery capabilities for application management. For information about creating and managing subscriptions, see Creating and managing subscriptions. See Application resource samples for sample YAML files.

Subscriptions are sets of definitions that identify Helm charts, deployables, and other Kubernetes resources within channels by using annotations, labels, and versions. Subscriptions can point to a channel or storage location for identifying new or updated deployables. The subscription operator can then download the subscribed Helm chart, deployable, or secret directly from the storage location to target managed clusters without checking the Hub cluster first. With a subscription, the subscription operator can monitor the channel for new or updated resources instead of the Hub cluster.

1.1.5. Placement rules

Placement rules (**placementrule.apps.open-cluster-management.io**) define the target clusters where deployables can be deployed. Use placement rules to help you facilitate the multi-cluster deployment of your deployables. For more information about creating and managing placement rules, see Creating and managing placement rules. See Placement rule samples for sample YAML files.

The custom resource definition (CRD) and controller for placement rules replaces the placement policies that were used for applications in previous versions of Red Hat Advanced Cluster Management for Kubernetes. Placement policies are still used for governance and risk policies.

Placement rules can be defined for subscriptions and for deployables. Define the placement rule at the subscription level for multi-cluster deployments. Define the placement rule for a specific deployable for single-cluster deployments or to override placement settings.

1.1.6. Application

Applications (**Application.app.k8s.io**) in Red Hat Advanced Cluster Management for Kubernetes are used for viewing the application components. See Application resource samples for sample YAML files.

1.1.6.1. Creating and managing channels

Create and use channels to supply application resources for the continuous integration and delivery capabilities of your Red Hat Advanced Cluster Management for Kubernetes applications.

Channels are custom resources that can help you manage application resources separately from the deployment of those resources onto managed clusters. Samples for all resources, including channels, are located in the Application resource samples documentation.

Channels (**channel.apps.open-cluster-management.io**) point to a physical place where resources are stored for deployment. They are created within a namespace on the hub cluster in which other resources related to the channel will also be stored.

There are four types of channels. Each channel differs based on the type of source location where resources are stored:

- Kubernetes namespace (**Namespace**): Uses deployables to store Kubernetes resource templates. A subscription for this type of channel retrieves and deploys the template. The namespaces that the channel monitors for new or updated deployables must be on the hub cluster.

- Object store (**ObjectBucket**): Stores Kubernetes resource YAML files. Each YAML file includes

the template portion for one resource, not the full deployable object. An object store can be populated with a deployable object that is on the hub cluster, or directly from a continuous integration pipeline, or by including the required YAML files into the object store.

- Helm repository (**HelmRepo**): Stores Helm charts. For information about how to structure your charts, see Helm documentation.

- Git repository (**Git**): Stores Kubernetes resources YAML files and unpackaged Helm charts. These resources do not need to be wrapped or represented as deployables. The channel controllers synchronize resources as deployables automatically.
  See the listed **Git** sources at are supported:

  - GitHub
  - GitLab
  - Bitbucket
  - Gogs (webhook is not supported)

Channels are created on the hub cluster by pointing to the location where resources are stored. Channels are defined with a **Channel** custom resource (CR).

Note: It is best practice to create each channel in a unique namespace. However, a Git channel can share a namespace with another type of channel, including Git, Helm, Kubernetes Namespace, and Object store.

The channel namespace is made available to a subscription operator on managed clusters when the cluster subscribes to the channel. The operator can then get the secrets to access the actual channel. The operator needs this access to check whether a deployable meets the channel requirements.

For more information application resources, see Application resources.

Learn more about channels, then see the following tasks:

- Creating channels
- Updating channel
- Deleting channel
- Managing deployments with channels
- Channel status and synchronization

1.1.6.1.1. Creating channels

1. Compose your channel definition YAML content. To create or update a channel resource, you must first compose the YAML file that defines the resource. For more information about the YAML structure, including the required fields, see Channel definition YAML structure .

2. Ensure that you create your channel in a unique namespace. All channels need an individual namespace, except Git channels, which can share a namespace with another channel.

3. Optional. If you are creating an **Namespace** type channel, create the namespace on your hub cluster.
   The channel type can be specified with the **spec.sourceNamespaces** and **spec.type** fields of a

channel spec. When a channel is created and pointing to a source, the channel controller maintains the promotion of the deployables at the source. The controller also synchronizes the source and channel namespaces.

You can define the namespace as part of your YAML definition or use the Kubernetes command line interface (**kubectl**) tool to create the namespace. To use the Kubernetes CLI tool, run the following command. Replace **namespace name** with the name for your new namespace:

```
kubectl create namespace <namespace name>`
```

4. Optional. If you need or want to use a Kubernetes Secret for authenticating the access to a repository or chart by the channel, create the Kubernetes Secret resource. You can use the Kubernetes command line interface (**kubectl**) tool to create the Kubernetes Secret by running the following command:

```
kubectl create secret <secret name> generic --from-literal=user --from-literal=password=
```

You can use a secret for authentication with only **HelmRepo**, **ObjectBucket**, and **Git** type channels. To associate a secret with a channel, include the **spec.secretRef.name** setting in your channel YAML definition.

5. Create the channel within Red Hat Advanced Cluster Management for Kubernetes. You can use the console, the Kubernetes command line interface (**kubectl**) tool, or REST API:

- To use the console,

  i. From the Navigation menu, click **Manage applications**. The **Overview** tab for all applications opens.

  ii. Click the **Resources** tab.

  iii. Scroll to the *Resource pipeline* section. From the list of buttons, find and click **Channel**. The *Create a Channel* editor is displayed.

  iv. Enter the YAML content to define your channel or directly update the default YAML template to meet your requirements.

  v. When you are finished, click **Save** to create the channel. Your new channel displays within the **Resource pipeline** for the corresponding applications.

Alternatively, you can select to create a channel when you are working with a specific application.

i. From the **Overview** tab for all applications, click the application from the **All applications** list. The **Overview** tab for that application opens.

ii. Click the **Resources** tab for that application.

iii. Scroll to the *Resource pipeline* section. From the list of buttons to the right of the resource summary cards, click **Channel**. The *Create a Channel* editor is displayed.

iv. Enter the YAML content to define your channel or directly update the default YAML template to meet your requirements.

v. When you are finished, click **Save** to create the channel. Your new channel displays within the **Resource pipeline** for the application.

- To use the Kubernetes CLI tool,

vi. Compose and save your channel YAML file with your preferred editing tool.

vii. Run the following command to apply your file to an apiserver. Replace **filename** with the name of your file:

```
kubectl apply -f filename.yaml
```

viii. Verify that your channel resource is created, by running the following command:

```
kubectl get Channel
```

Ensure that your new channel is listed in the resulting output. ** To use REST API, use the APIs.

1.1.6.1.2. Updating channel

1. Compose the definition updates for your channel. For more information about the YAML structure, including the required fields, see Channel definition YAML structure .
2. Update the definition. You can use the console, the Kubernetes command line interface (**kubectl**) tool, or REST API:

- To use the console,

i. Open the console.

ii. From the Navigation menu, click **Manage applications**. The **Overview** tab for all applications opens.

iii. Click the **Resources** tab.

iv. Scroll down the page to **Resource pipeline** section. Click the **YAML** edit icon for channel that you want to update. The **Edit channel** window opens.

v. Edit the YAML for the channel.

vi. When you are finished, click **Save** to update the channel.

You can also use the console search to find and edit a channel:

i. From the Navigation menu, click **Search**.

ii. Within the search box, filter by **kind:channel** to view all channels.

iii. Within the list of all channels, click the channel that you want to update. The YAML for the channel is displayed.

iv. Click **Edit** to enable editing the YAML content.

v. When you are finished your edits, click **Save**. Your changes are saved and applied automatically.

- To use the Kubernetes CLI tool, the steps are the same as for creating a channel.

```
To use REST API, use the link:../apis#apis[APIs].
```

1.1.6.1.2.1. Deleting channel

To delete a channel, you can use the console, the Kubernetes command line interface (**kubectl**) tool, or REST API.

To use the console, complete the following procedure:

1. From the navigation, click on **Manage applications**.
2. Click the **Resources** tab.
3. Find your channel resource card that you want to delete.
4. Click **Options** for more actions.
5. Click **Delete channel**
6. Verify when the list of all channels is refreshed, the channel is no longer displayed.

To use the Kubernetes CLI tool:

1. Run the following command to delete the channel from a target namespace.
2. Replace **name** and **namespace** with the name of your channel and your target namespace:

```
kubectl delete Channel <name> -n <namespace>
```

3. Verify that your channel is deleted by running the following command:

```
kubectl get Channel <name>
```

To use REST API, use the APIs.

1.1.6.1.3. Managing deployments with channels

You can use channels and subscriptions to manage the continuous delivery of deployables, such as Helm charts and Kubernetes deployable objects, to your managed clusters or other namespaces.

When a channel points to a Helm repository, the channel operator creates a deployable to represent each Helm release that is found in the repository.

For Kubernetes deployable objects, you can add the objects to a channel by wrapping them as deployables.

Within the deployable definition you can directly specify a channel where the deployable is to be promoted. You can also specify the required Kubernetes labels for the deployable to match the gate requirements for a channel to have the deployable automatically added to a channel. When the channel controller detects that the deployable includes the required Kubernetes labels, the controller promotes the deployable to the channel.

When creating a **Namespace** and **ObjectBucket** channel, you can set the channel gate requirements within the **spec.gates** section of the channel definition. These requirements are Kubernetes annotations that a deployable must include before the deployable can be promoted to the channel. For instance, you can specify annotations for development approvals, test and quality assurance approval, and to indicate a deployable is ready for deployment to a production environment cluster. These gate requirements can be any field and value, such as the source namespace, package name, labels, and

annotations. A deployable can be promoted in a channel only when the deployable definition includes the matching fields and values. When a deployable meets the defined requirements, the deployable is automatically promoted to the channel and deployed to any managed clusters that subscribe to the channel. Gate requirements do not apply for **HelmRepo** and **Git** channel types.

For channels that do not include gate requirements, the channel controller promotes the latest version of a deployable to the channel.

Clusters can subscribe to channels for identifying the deployables to deploy to each cluster. Deployables that are promoted to a channel can be accessed by only the subscriptions for that channel. When channels and subscriptions exist, the channels and subscriptions work together to retrieve deployables from the channel source and place the deployables at the destination. The destination is typically a managed cluster, which is abstracted as a namespace. A managed cluster or namespace can subscribe to multiple channels for identifying the deployables to deploy to the cluster. Channels ensure that the correct deployable is available for retrieval. Subscriptions ensure that the deployable is retrieved and placed on the destination namespaces or clusters. For retrieving a deployable, the subscription operator checks the annotation limits and determines whether to retrieve and apply the deployable to the managed cluster.

1.1.6.1.4. Channel status and synchronization

Channels do not have a status, while a subscription that subscribes to a channel does have a status. The status for a subscription reports whether the subscription is successfully propagated on the hub cluster, and whether the subscription successfully created or applied a deployable template or created the helmRelease CR. The status for the deployables that are deployed through the use of channels and subscriptions are reported separately from the subscription.

Since channels do not have a status, the resources that are promoted to a channel namespace are not always synchronized with the actual channel storage.

When a deployable is included within a channel or updated, the subscription operator for the channel detects the new or updated deployable automatically. You do not need to issue any notification from the hub cluster to the target managed clusters.

For a subscription to a namespace type channel, resources are synchronized for a managed cluster only while the cluster can access the channel namespace. Since the channel source exists on the hub cluster, the subscription can only pull the resources from the source when access to the hub cluster exists.

For subscriptions to Helm repository and object store type channels, the subscription watches the channel source repositories for new or updated Helm charts or deployables. The subscriptions do not need to communicate with the hub cluster unless the source repository is on the hub cluster. If subscriptions are set up to pull the latest version for a Helm release or deployable object, and new versions are included in the repository types, the subscriptions can retrieve the versions.

1.1.6.1.5. Managing secrets

You can create Kubernetes secret resources to store authorization credentials and other sensitive information for your subscriptions and application components.

Samples for resources, including secrets, are located in the Application resource samples documentation.

1.1.6.1.5.1. Kubernetes secrets

Secrets (**Secret**) are Kubernetes resources that you can use to store authorization and other sensitive information, such as passwords, OAuth tokens, and SSH keys.

With secrets, you can separate and store information securely, while using subscriptions to send information where its needed.

For example, you can use image pull secrets, which include credentials for a private Docker image registry that your deployment resource references.

By including image pull secrets in a managed cluster namespace where a subscription to a Helm chart or deployed pod is included, the Helm chart or pod can then use the pull secret within the namespaced scope.

When you create and include a secret within a channel for deployment to managed clusters, that secret can be deployed to only the same namespace on the target managed clusters as the subscription (the subscription uses the same namespace on the hub and managed cluster).

All secret data is encrypted end-to-end. On the hub cluster and the managed cluster, Kubernetes encrypts its secrets at rest. During transport, TLS encryption is used.

1. Create a namespace where the *channel* and *secret* resources will be stored on the hub cluster. **Required access:** Cluster administrator. You only need to complete this step one time by running the following command:

   ```
   oc new-project SECRET_NAMESPACE
   ```

   Alternately, you can run **oc apply -f FILENAME.yaml** to apply the following YAML sample. Copy and paste:

   ```
   apiVersion: v1
   kind: Namespace
   metadata:
     name: SECRET_NAMESPACE
   ```

2. Create a channel resource. The secret namespace needs to also contain a channel resource. Run the **oc apply -f FILENAME.yaml** command to apply the following sample YAML file:

   ```
   apiVersion: apps.open-cluster-management.io/v1
   kind: Channel
   metadata:
     name: CHANNEL_NAME
     namespace: SECRET_NAMESPACE
   spec:
     type: Namespace
     pathname: SECRET_NAMESPACE
   ```

   The following YAML definition table contains values for the keys need to be provided:

Key	Value
name	A unique name for the channel, constrained to the dns format.
namespace	Optional. If not provided, the channel is created in the environments current namespace. Must be the same as the secret you want to subscribe.

Key	Value
pathname	Should be the same value as the namespace.

3. Create your secrets by running the **oc apply -f FILENAME.yaml** command to apply the following sample YAML file. You can create as many secrets as you need:

```
apiVersion: v1
kind: Secret
metadata:
  annotations:
     apps.open-cluster-management.io/deployables: "true"
  name: SECRET_NAME
  namespace: SECRET_NAMESPACE
data:
```

The following YAML definition table contains values for the keys need to be provided:

Key	Value
name	A unique name, constrained to the dns format
namespace	Optional. If not provided, the secret is created in the environments current namespace
data	This is either YAML or a large string block that you want to store securely

4. Create your subscriptions. The subscription is used to subscribe one or more secrets from the channel and needs to be created in its own namespace. This namespace is propagated to the managed cluster, where the secret is created.
The following sample defines a local subscription placement:

```
placement:
  local: true
```

The following sample defines a managed cluster (remote) subscription placement:

```
placement:
  placementRef:
    name: PLACEMENT_NAME
    kind: PlacementRule
```

The following sample defines a subscription:

```
apiVersion: apps.open-cluster-management.io/v1
kind: Subscription
metadata:
  name: SUBSCRIPTION_NAME
  namespace: SUBSCRIPTION_NAMESPACE
```

```
spec:
  channel: CHANNEL_NAMESPACE/CHANNEL_NAME
  placement:
   local: true
```

5. To verify that you all secrets in the channel are applied, run the following command:

```
oc -n SUBSCRIPTION_NAMESPACE describe appsub SUBSCRIPTION_NAME
```

1.1.6.2. Creating and managing subscriptions

You can create and manage subscriptions to identify, retrieve, and deploy new and updated resources to managed clusters. By using subscriptions, you can improve the continuous delivery capabilities of your application management. Samples for all resources, including subscriptions, are located in the Application resource samples documentation.

Learn more about subscriptions, then see the following tasks:

- Creating a subscription
- Matching a subscription to an application
- Updating a subscription
- Scheduling a deployment
- Subscribing Git resources
- Configuring package overrides
- Deleting a subscription

Subscriptions (**subscription.apps.open-cluster-management.io**) are Kubernetes resources that serve as sets of definitions for identifying Kubernetes resources (in Git, Objectstores, or hub cluster deployables), and Helm charts within channels by using annotations, labels, and versions.

Subscription resources can point to a channel for identifying new and updated Helm charts or Kubernetes resources for deployment. The subscription operator then watches the channel for new and updated charts and deployables.

When a new or updated Helm chart or Kubernetes resource is detected, the subscription operator downloads the Helm release version for the specified Helm chart version or the specified Kubernetes resource. The subscription operator can download these objects directly, or as deployables, from the storage location to target managed clusters without checking the hub cluster first.

Subscriptions can filter the deployables that are promoted to a channel to select specific deployables. For instance, the subscription can filter the deployables to select a specific deployable version. For this case, the subscription operator checks the **version** parameter to identify the deployable version to select.

1.1.6.2.1. Creating a subscription

1. Compose the definition YAML content for your subscription. For more information about YAML structure and samples, see Application resource samples documentation.

2. Create the subscription within Red Hat Advanced Cluster Management for Kubernetes. You can use the console, the Kubernetes CLI (**kubectl**) tool, or REST API:

   - To use the console:

     i. Open the console.

     ii. From the Navigation menu, click **Manage applications**. The **Overview** tab for all applications opens.

     iii. Click the **Resources** tab.

     iv. Scroll to the *Resource pipeline* section. From the list of buttons to the right of the resource summary cards, click **Susbcription**. The *Create a Susbcription* editor is displayed.

     v. Enter the YAML content to define your subscription or directly update the default YAML template to meet your requirements.

     vi. When you are finished, click **Save** to create the subscription. Your new subscription displays within the **Resource pipeline** for the corresponding applications and channels.

     Alternatively, you can select to create a subscription when you are working with a specific application.

     i. From the **Overview** tab for all applications, click the application from the **All applications** list. The **Overview** tab for that application opens.

     ii. Click the **Resources** tab for that application.

     iii. Scroll to the *Resource pipeline* section. From the list of buttons to the right of the resource summary cards, click **Susbcription**. The *Create a Susbcription* editor is displayed.

     iv. Enter the YAML content to define your subscription or directly update the default YAML template to meet your requirements.

     v. When you are finished, click **Save** to create the subscription. Your new subscription displays within the **Resource pipeline** for the corresponding applications and channels.

   NOTE: The subscription might not initially display in your resource pipeline for any application. For the subscription to be associated with an application, you need to include the appropriate values in your subscription definition to match the required values that are defined by the application definition. For more information, see Matching a subscription to an application .

   - To use the Kubernetes CLI tool:

     i. Run the following command to apply your file to an apiserver. Replace **filename** with the name of your file:

     ```
     kubectl apply -f filename.yaml
     ```

     ii. Verify that your subscription resource is created, by running the following command:

     ```
     kubectl get appsub
     ```

     Ensure that your new subscription is listed in the resulting output.

- To use REST API, use the APIs.

After your subscription is created, your subscription can have one of the following statuses:

- **Subscribed** (managed clusters only)
  The subscription has successfully subscribed to the specified resources and the resources are deployed.

- **Propagated** (hub cluster only)
  The subsequent subscriptions based on your specified **spec.placement** setting are generated for deployment to the appropriate managed clusters, if any. However, the subscriptions might not actually be deployed.

- **Failed** The subscription failed to subscribe to the specified resources.

- **No status**
  The subscription operator is either not online, or the **spec.placement** setting for the subscription is missing.

If you created the subscription for a multi-cluster environment, the subscription is created on the Hub cluster. Depending on your **spec.placement** settings, the subsequent placement of your subscription on other clusters is different. If you include more than one placement setting within the **spec.placement** section, the subscription operator uses the following priority to select the placement setting to use:

- **spec.placement.placementRef**
  The subscription is placed on the clusters that are identified by the specified **PlacementRule** resource.

- **spec.placement.clusters**
  The subscription is placed on the clusters that are specified as the value for this field.

- **spec.placement.clusterSelector**
  The subscription is placed on the clusters that match the specified label selector that is defined as the value for this field.

If you created your subscription on a stand-alone cluster or a cluster that you want to manage directly, the subscription is created on only that cluster. Depending on the value that you set for the **spec.placement.local** field within the subscription definition, the subsequent behavior of your subscription is different.

- **true**
  The subscription synchronizes with the specified channel that is local to that cluster.

- **false**
  The subscription does not subscribe to any resources from the specified channel.

1.1.6.2.2. Matching a subscription to an application

To associate a subscription with an application, both the subscription and application must be in the same namespace so that the subscription can retrieve Helm charts, deployables, or other resources from a channel.

Within the application resource definition, the definition must include **spec.componentKinds** settings to indicate that the application uses a subscription. The definition must also include **spec.selector** settings to define the labels (**matchLabels**) or expressions (**matchExpressions**) to use to match the application with the subscription.

Within the subscription resource definition, the definition must include the required values to match the labels or expressions that are defined by the application.

When the subscription is associated with an application, the subscription uses the **spec.placement** settings for the subscription or deployable to deploy any subscribed charts, deployables or other Kubernetes resources for the application.

For more information about the resource definition for an application, see Creating and managing application resources.

1.1.6.2.3. Updating a subscription

1. Compose the definition YAML content for your subscription.
2. Create the subscription within Red Hat Advanced Cluster Management for Kubernetes. You can use the console, the Kubernetes CLI (**kubectl**) tool, or REST API:
   - To use the console:
     i. Open the console.
     ii. From the Navigation menu, click **Manage applications**. The **Overview** tab for all applications opens.
     iii. Click the **Resources** tab.
     iv. Scroll down the page to **Resource pipeline** section. Expand the row for the application that uses the subscription that you want to edit.
     v. For the subscription that you want to update, click the **Edit** icon for the YAML. The **Edit subscription** window opens.
     vi. Edit the YAML.
     vii. When you are finished, click **Save** to update the subscription.

     Alternatively, you can select to update the subscription when you are working with a specific application.
     i. From the **Overview** tab for all applications, click the application from the **All applications** list. The **Overview** tab for that application opens.
     ii. Click the **Resources** tab for that application.
     iii. Scroll down the page to **Resource pipeline** section.
     iv. For the subscription that you want to update, click the **Edit** icon for the YAML. The **Edit subscription** window opens.
     v. Edit the YAML.
     vi. When you are finished, click **Save** to update the subscription.

   You can also use the console search to find and edit a subscription:
   i. Click the **Search** icon in the Header.
   ii. Within the search box, filter by **kind:subscription** to view all subscriptions.

iii. Within the list of all subscriptions, click the subscription that you want to update. The YAML for the subscription is displayed.

iv. Click **Edit** to enable editing the YAML content.

v. When you are finished your edits, click **Save**. Your changes are saved and applied automatically.

- To use the Kubernetes CLI tool, the steps are the same as for creating a subscription.

> To use REST API, use the link:../apis#apis[APIs]

1.1.6.2.4. Scheduling a deployment

If you need to deploy new or change Helm charts or other resources during only specific times, you can define subscriptions for those resources to begin deployments during only those specific times. Alternatively, you can restrict deployments from beginning during specific time windows, such as to avoid unexpected deployments during peak business hours.

For instance, you can define time windows between 10:00 PM and 11:00 PM each Friday to serve as scheduled maintenance windows for applying patches or other application updates to your clusters.

Alternatively, you can restrict or block deployments from beginning during specific time windows, such as to avoid unexpected deployments during peak business hours. For instance, to avoid peak hours you can define a time window for a subscription to avoid beginning deployments between 8:00 AM and 8:00 PM.

By defining time windows for your subscriptions, you can coordinate updates for all of your applications and clusters. For instance, you can define subscriptions to deploy only new application resources between 6:01 PM and 11:59 PM and define other subscriptions to deploy only updated versions of existing resources between 12:00 AM to 7:59 AM.

When a time window is defined for a subscription, the time ranges when a subscription is active changes. As part of defining a time window, you can define the subscription to be *active* or *blocked* during that window. The deployment of new or changed resources begins only when the subscription is active. Regardless of whether a subscription is active or blocked, the subscription continues to monitor for any new or changed resource. The active and blocked setting affects only deployments.

When a new or changed resource is detected, the time window definition determines the next action for the subscription.

- For subscriptions to **HelmRepo**, **ObjectBucket**, and **Git** type channels:
- If the resource is detected during the time range when the subscription is active, the resource deployment begins.
- If the resource is detected outside the time range when the subscription is blocked from running deployments, the request to deploy the resource is cached. When the next time range that the subscription is active occurs, the cached requests are applied and any related deployments begin.
- For subscriptions to **Namespace** type channels:
- When a subscription becomes active, the subscription synchronizes with the channel and begins the deployment for the latest version of any resources that need to be deployed.

- When the subscription is blocked, the subscription is not synchronized with the channel for deploying resources.

If a deployment begins during a defined time window and is running when the defined end of the time window elapses, the deployment continues to run to completion.

To define a time window for a subscription, you need to add the required fields and values to the subscription resource definition YAML.

- As part of defining a time window, you can define the days and hours for the time window.
- You can also define the time window type, which determines whether the time window when deployments can begin occurs during, or outside, the defined timeframe.
- If the time window type is **active**, deployments can begin only during the defined timeframe. You can use this setting when you want deployments to occur within only specific maintenance windows.
- If the time window type is **block**, deployments cannot begin during the defined timeframe, but can begin at any other time. You can use this setting when you have critical updates that are required, but still need to avoid deployments during specific time ranges. For instance, you can use this type to define a time window to allow security-related updates to be applied at any time except between 10:00 AM and 2:00 PM.
- You can define multiple time windows for a subscription, such as to define a time window every Monday and Wednesday.

1.1.6.2.4.1. Subscribing Git resources

You can subscribe Git resources to multiple namespaces. By default, when you subscribe to an application, the application is deployed into that subscription namespace. To apply these resources to namespaces outside of that subscription namespace, see the following procedure:

Required access: Cluster administrator

1. From the console, log in to your hub cluster.
2. Create one or more users.
   These users represent administrators for the **app.open-cluster-management.io/subscription** application. With OpenShift Container Platform, you can group these users to represent a subscription administrative group, which is demonstrated later in this topic.
3. From the terminal, log in to your Red Hat Advanced Cluster Management hub cluster again.
4. Add the following subjects into **open-cluster-management:subscription-admin** ClusterRoleBinding with the following command:

   ```
   oc edit clusterrolebinding open-cluster-management:subscription-admin
   ```

   Note: open-cluster-management:subscription-admin ClusterRoleBinding has no subject initially.

   Your subjects might display as the following example:

   ```
   subjects:
   - apiGroup: rbac.authorization.k8s.io
   ```

```
  kind: User
  name: example-name
- apiGroup: rbac.authorization.k8s.io
  kind: Group
  name: example-group-name
```

To verify, see an example where you are logged in as a subscription administrator. In the example, you subscribe the sample resource YAML file from a Git repository, follow the previous procedure, then see updated Configmap settings. The example file contains subscriptions that are located within the following different namespaces:

- Configmap **test-configmap-1** gets created in **multins** namespace.
- Configmap **test-configmap-2** gets created in **default** namespace.
- Configmap **test-configmap-3** gets created in the **subscription** namespace.

```
---
apiVersion: v1
kind: Namespace
metadata:
  name: multins
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: test-configmap-1
  namespace: multins
data:
  path: resource1
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: test-configmap-2
  namespace: default
data:
  path: resource2
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: test-configmap-3
data:
  path: resource3
```

1.1.6.2.5. Configuring package overrides

Configure package overrides for a subscription override value for the Helm chart or Kubernetes resource that is subscribed to by the subscription.

To configure a package override, specify the field within the Kubernetes resource spec to override as the value for the **path** field. Specify the replacement value as the value for the **value** field.

For example, if you need to override the values field within the spec for a Helm release for a subscribed Helm chart, you need to set the value for the **path** field in your subscription definition to **spec**.

```
packageOverrides:
- packageName: nginx-ingress
  packageOverrides:
  - path: spec
    value: my-override-values
```

The contents for the **value** field are used to override the values within the **spec** field of the **HelmRelease** spec.

- For a Helm release, override values for the **spec** field are merged into the Helm release **values.yaml** file to override the existing values. This file is used to retrieve the configurable variables for the Helm release.
- If you need to override the release name for a Helm release, include the **packageOverride** section within your definition. Define the **packageAlias** for the Helm release by including the following fields:
  - **packageName** to identify the Helm chart.
  - **packageAlias** to indicate that you are overriding the release name.

  By default, if no Helm release name is specified, the Helm chart name is used to identify the release. In some cases, such as when there are multiple releases subscribed to the same chart, conflicts can occur. The release name must be unique among the subscriptions within a namespace. If the release name for a subscription that you are creating is not unique, an error occurs. You must set a different release name for your subscription by defining a **packageOverride**. If you want to change the name within an existing subscription, you must first delete that subscription and then recreate the subscription with the preferred release name.

  +

  ```
  packageOverrides:
  - packageName: nginx-ingress
    packageAlias: my-helm-release-name
  ```

1.1.6.2.6. Deleting a subscription

To delete a subscription, you can use the console, the Kubernetes command line interface (**kubectl**) tool, or REST API.

To use the console, complete the following procedure:

1. From the navigation, click on **Manage applications**.
2. Click the **Resources** tab.
3. Find your subscription resource card that you want to delete.
4. Click **Options** for more actions.
5. Click **Delete subscription**
6. Verify when the list of all subscriptions is refreshed that the subscription you deleted is no longer displayed.

To use the Kubernetes CLI tool:

1. Run the following command to delete the subscription from a target namespace.
2. Replace **name** and **namespace** with the name of your subscription and your target namespace:

```
kubectl delete appsub <name> -n <namespace>
```

3. Verify that your subscription is deleted by running the following command:

```
kubectl get appsub <name>
```

To use REST API, use the APIs.

1.1.6.3. Creating and managing placement rules

You can create and manage placement rules to define where and how Helm charts and deployables are deployed. Placement rules help you facilitate multi-cluster deployments of your deployables. Samples for all resources, including placement rules, are located in the Application resource samplesdocumentation.

- Create a placement rule
- Assign a placement rule
- View placement status
- Update a placement rule
- Delete a placement rule

The custom resource definition (CRD) and controller for placement rules replaces the placement policies that were used for applications in previous versions of Red Hat Advanced Cluster Management for Kubernetes. Placement policies are still used for governance and risk policies.

Placement rules can be defined for subscriptions and for deployables. Define the placement rule at the subscription level for multi-cluster deployments. Define the placement rule for a specific deployable for single-cluster deployments or to override placement settings.

1.1.6.3.1. Create a placement rule

Placement rules can be defined for subscriptions and for deployables. Define the placement rule at the subscription level for multi-cluster deployments. Define the placement rule for a specific deployable for single-cluster deployments or to override placement settings.

Prerequisite: Be sure the **klusterlet-addon-appmgr** pod is running. You can run **oc get pods -n open-cluster-management-agent-addon** to check for pods.

1. Compose the definition YAML content for your placement rule. Samples for all resources, including placement rules, are located in the Application resource samplesdocumentation.
2. Create the placement rule within Red Hat Advanced Cluster Management for Kubernetes. You can define a placement rule as a separate resource, or define the rule within the definition for a deployable or subscription.

As a best practice, define placement rules as a separate resource when the rule might need to be referenced by multiple resources.

+ To create a placement rule as a separate resource, you can use the console, the Kubernetes CLI (**kubectl**) tool, or REST API:

- To use the console:
  i. Open the console.
  ii. From the Navigation menu, click **Manage applications**. The **Overview** tab for all applications opens.
  iii. Click the **Resources** tab.
  iv. Scroll to the *Resource pipeline* section. From the list of buttons, find and click **Placement Rule**. The *Create a placement rule* editor is displayed.
  v. Enter the YAML content to define your placement rule or directly update the default YAML template to meet your requirements.
  vi. When you are finished adding or editing the YAML, click **Save** to create the placement rule.
  vii. From the list of resource summary cards, click the *PLACEMENT RULES* card. The *Search* dashboard is displayed and lists all placement rules that are used by the applications that are selected on the *Applications* dashboard. Your new placement rule does not display in this list until the rule is referenced by a subscription that is used by an application. To verify that your new placement rule is created, enter **kind:placementrule** in the search box and run your search. Ensure that your new rule is displayed in the search results list.
- To use the Kubernetes CLI tool:
  i. Run the following command to apply your file to an apiserver. Replace **filename** with the name of your file:

```
kubectl apply -f filename.yaml
```

  ii. Verify that your placement rule is created, by running the following command:

```
kubectl get PlacementRule
```

  Ensure that your new placement rule is listed in the resulting output.
- To use REST API, use the APIs.

1.1.6.3.2. Assign a placement rule

You can assign a placement rule to a deployable or subscription. To assign a placement rule, you need to update the spec for the deployable or subscription to reference the placement rule.

Include the following **placement** fields in your deployable or subscription spec with the values to reference the placement rule:

```
placement:
  placementRef:
    name:
    kind: PlacementRule
```

Include the name of your placement rule as the value for the **name** field.

When a placement rule is assigned to a subscription, you can view the assignment on the Applications dashboard within the console with the following procedure:

1. Open the console.
2. From the Navigation menu, click **Manage applications**. The **Overview** tab for all applications opens.
3. Click the **Resources** tab.
4. Scroll to the **Resource pipeline** section. Within the table that lists your applications, expand the row for the application that includes the subscription that is assigned the placement rule.
5. From the expanded view for the application, you can see the available subscriptions for each channel. The details for each subscription include any assigned placement rule. If needed, you can select to view or edit the YAML for the placement rule, subscription, and channel from this resource pipeline table.

1.1.6.3.3. View placement rule status

When a placement rule is created and in use, you can view the status details for the rule. This status is appended to the YAML definition for a placement rule and indicates the target clusters where the rule is used for placing deployables.

To view the status fields for a placement rule, you can use the console, the Kubernetes command line interface (**kubectl**) tool, or REST API.

- To use the console,
  a. Open the console.
  b. Click the **Search** icon in the Header.
  c. Within the search box, filter by **kind:placementrule** to view all placement rules.
  d. Within the list of all placement rules, click the placement rule that you want review. The YAML for that rule is displayed.
  e. Review the fields and values within the **status** section of the YAML content.
- To use the Kubernetes CLI tool, run the following command. Replace **name** and **namespace** with the name of the placement rule and the target namespace:
  a. Run the following command

```
kubectl get PlacementRule <name> -n <namespace>
```

  b. Review the fields and values within the **status** section of the YAML content.

To use REST API, use the APIs

1.1.6.3.4. Update a placement rule

To update a placement rule that is a separate resource, you can use the console, the Kubernetes command line interface (**kubectl**) tool, or REST API.

- To use the console to edit a placement rule, complete the following steps:

a. Open the console.

b. Click the **Search** icon in the Header.

c. Within the search box, filter by **kind:placementrule** to view all placement rules.

d. Within the list of all placement rules, click the placement rule that you want to update. The YAML for the rule is displayed.

e. Click **Edit** to enable editing the YAML content.

f. When you are finished your edits, click **Save**. Your changes are saved and applied automatically.

Alternatively, you can select to edit the YAML from the Applications dashboard resource pipeline table.

a. From the Navigation menu, click **Manage applications**. The **Overview** tab for all applications opens.

b. Click the **Resources** tab.

c. Scroll to the **Resource pipeline** section. Within the table that lists your applications, expand the row for the application that includes the subscription that is assigned the placement rule.

d. From the expanded view for the application, you can see the available subscriptions for each channel. The details for each subscription include any assigned placement rule. Click the link for the placement rule to open the *Edit placement rule* editor. The YAML for the rule is displayed.

e. When you are finished your edits, click **Save**. Your changes are saved and applied automatically.

- To use the Kubernetes CLI tool, the steps are the same as for creating a placement rule.

To use REST API, use the APIs.

To update a placement rule that is defined within the definition for a deployable or subscriptions, the steps are the same as for updating that resource.

1.1.6.3.5. Delete a placement rule

To delete a placement rule that is a separate resource, you can use the console, the Kubernetes command line interface (**kubectl**) tool, or REST API.

To use the console, complete the following procedure:

1. From the navigation, click on **Manage applications**.
2. Click the **Resources** tab.
3. Find your subscription resource card for the placement rule that you want to delete.
4. Click **Options** for more actions.
5. Click **Delete placement rule**

6. Verify when the list of all subscriptions is refreshed that the placement rule is no longer displayed.

To use the Kubernetes CLI tool, complete the following steps:

1. Run the following command to delete the placement rule from a target namespace.
2. Replace **name** and **namespace** with the name of your placement rule and your target namespace:

```
kubectl delete PlacementRule <name> -n <namespace>
```

3. Verify that your placement rule resource is deleted by running the following command:

```
kubectl get PlacementRule <name>
```

To use REST API, use the APIs.

1.1.6.4. Creating and managing application resources

Create application resources to group and view the application components that make up your overall Red Hat Advanced Cluster Management for Kubernetes multi-cluster applications. Samples for all resources are located in the Application resource samples documentation.

- Create an application
- Matching a subscription to an application
- Update an application
- Delete an application

1.1.6.4.1. Create an application

1. Compose your application definition YAML content. To create or update an application resource, you must first compose the YAML file that defines the resource.
2. Create the application within Red Hat Advanced Cluster Management for Kubernetes. You can use the console, the Kubernetes command line interface (**kubectl**) tool, or REST API:
   - To use the console,
     i. Open the console.
     ii. From the Navigation menu, click **Manage applications**. The **Overview** tab for all applications opens.
     iii. Click **New application**. The *Create application* window opens.
     iv. Update or replace the default YAML content within the editor to define your application.
     v. When you are finished adding and editing the YAML, click **Save** to create the application. Your new application displays within the list of applications on the **Overview** tab.

- To use the Kubernetes CLI tool,

  i. Compose and save your application YAML file with your preferred editing tool.

  ii. Run the following command to apply your file to an apiserver. Replace **filename** with the name of your file:

  ```
  kubectl apply -f filename.yaml
  ```

  iii. Verify that your application resource is created by running the following command:

  ```
  kubectl get Application
  ```

  Ensure that your new application is listed in the resulting output.

- To use REST API, use the APIs.

1.1.6.4.2. Update an application

1. Compose your application definition updates. Samples for all resources are located in the Application resource samples documentation.
2. Update the application definition. You can use the console, the Kubernetes command line interface (**kubectl**) tool, or REST API:

   - To use the console, complete the following steps:

     i. Open the console.

     ii. From the Navigation menu, click **Manage applications**. The **Overview** tab for all applications opens.

     iii. Within the list of all applications, find the row for the application that you want to update. For that row expand the *Options* menu and click **Edit application**. The *Edit application* window opens.

     iv. Update the YAML content for the application.

     v. When you are finished your edits, click **Close editor**. Your changes are saved and applied automatically.

   You can also use the console search to find and edit an application:

   i. Click the *Search* icon in the Header to open the *Search* page.

   ii. Within the search box, filter by **kind:application** to view all applications.

   iii. Within the list of all applications, click the application that you want to update. The *Overview* page for that application opens.

   iv. Click **Edit app**. The *Edit application* window opens.

   v. Update the YAML content for the application.

   vi. When you are finished your edits, click **Close editor**. Your changes are saved and applied automatically.

- To use the Kubernetes CLI tool, the steps are the same as for creating an application.

To use REST API, use the APIs.

1.1.6.4.3. Delete an application

To delete an application, you can use the console, the Kubernetes command line interface (**kubectl**) tool, or REST API:

To use the console, complete the following steps:

1. From the Navigation menu, click **Manage applications**. The **Overview** tab for all applications opens.
2. Within the list of all applications, find the row for the application that you want to delete.
3. On that row, expand the *Options* menu and click **Delete application**. A confirmation window opens.
4. Confirm the removal to delete the application.
5. When the list of all applications is refreshed, the application is no longer included.

To use the Kubernetes CLI tool, complete the following steps:

1. Run the following command to delete the application from a target namespace.
2. Replace **name** and **namespace** with the name of your application and your target namespace:

```
kubectl delete Application <name> -n <namespace>
```

3. Verify that your application resource is deleted by running the following command:

```
kubectl get Application <name>
```

To use REST API, use the APIs.

1.1.6.4.4. Deploying by using application resources

To set up and use channels, subscriptions, and placement rules for deployments, complete the following procedure:

1. If the channel to represent the object store, Kubernetes namespace, or Helm repository does not exist, create a channel.
   Ensure that you define any labels or annotations that deployables need before they are promoted to the channel. For more information, see Creating and managing channels.
2. If your target cluster or clusters are not subscribed to the channel, create the subscription. For more information, see Creating and managing subscriptions.
3. Define the placement rule for the deployables that are to be deployed from the channel. For more information, see Creating and managing placement rules.

A placement rule can be defined for a subscription and for deployables. **Define the placement rule for a subscription to have the rule apply to multiple deployables and clusters.** Define the placement rule for a deployable to prevent the rule from applyin to all deployables for the subscription, or when you

want to override a subscription-level placement rule. ** If you are referencing a placement rule, you can also include override settings for a deployable to override some placement rule values with values specific to the deployable. . Optional. If you created the placement rule as a stand-alone resource, edit the definition for your subscription or deployables to reference your placement rule. . Edit the definition for your deployables and your channel to ensure that deployables are promoted to the channel. For more information, see the *Promoting a deployable to a channel* section. . Use the console to monitor the status of the deployment to the target cluster or clusters for the channel.

1.1.6.4.5. Promoting a deployable to a channel

To promote a deployable, which is a *resource template*, to a channel, you can use any of the following methods:

- Point the deployable to a specific channel by configuring the **spec.channels** field within the deployable definition with the correct annotations to identify the channel.
  The **spec.channels** parameter is available for deployable (**deployable.apps.open-cluster-management.io**) resources to identify the channels where the deployable is to be promoted. The following example shows a deployable that defines the channels where the deployable is to be included:

```
apiVersion: apps.open-cluster-management.io/v1
kind: Deployable
metadata:
  name: deployable1
  namespace: default
  spec:
    template:
    placement:
    overrides:
    dependencies:
    channels:
    - mychannel
    - otherchannel
```

- Update the channel definition to identify the deployables to include in the channel. Use the **spec.package** and **spec.packageFilter** fields to specify the deployables.
  As an example, the following channel automatically pulls the most recent, or latest, **nginx** chart when a new chart is published to the source Helm repository for the channel. The chart deployable must have a matching version **1.x** to be promoted to the channel.

```
apiVersion: apps.open-cluster-management.io/v1
kind: Channel
metadata:
  name: predev-ch
  namespace: ns-ch
  labels:
    app: nginx-app-details
spec:
  type: HelmRepo
  pathname: https://kubernetes-charts.storage.googleapis.com/
---
apiVersion: apps.open-cluster-management.io/v1
kind: Subscription
metadata:
  name: nginx
```

```
    namespace: ns-sub-1
    labels:
      app: nginx-app-details
  spec:
    channel: ns-ch/predev-ch
    name: nginx-ingress
    packageFilter:
      version: "1.36.x"
    placement:
      placementRef:
        kind: PlacementRule
        name: towhichcluster
```

- Update the subscription definition to identify the deployables. The configuration for promoting a deployable to a channel can also be specified within the subscription definition.

The following example subscription indicates that the most recent **nginx** version **1.x** chart is to be promoted through the channel for deployment with the subscription.

```
```yaml
apiVersion: apps.open-cluster-management.io/v1
kind: Subscription
metadata:
  name: mydevsub
  namespace: myspace
spec:
 source: https://kubernetes-charts.storage.googleapis.com/
 package: nginx
 packageFilter:
   version: 1.x
 placement:
   clusters:
   - name: mydevcluster1
```
```

- Update the channel definition to specify channel gate requirements, and update the definitions for your deployables to include the fields and values to match the gate requirements. Channel gate requirements are defined within the **spec.gate** section of a channel definition. If the deployable has the fields to match the channel **spec.gate** values, the deployable is promoted to the channel. In this case, the deployable does not need to point to a specific channel with the **spec.channels** field.

  In the previous example, **packageFilter.version: "1.36.x"** indicates the specific **nginx** version **1.36.x** chart is promoted through the channel for deployment with the subscription.

1.1.6.4.5.1. Deploying with a percentage roll out

If you want to roll out a deployment to your target managed clusters instead of deploying to all target clusters, you can configure the deployment of a deployable or chart to only a percentage of your managed clusters at a time.

For instance, you might want to roll out a deployment when you need to deploy an update but you do not want to affect all clusters at once. When the deployment is successful on a cluster, the deployment is rolled out to another cluster.

1.1.6.5. Application resource samples

Within Red Hat Advanced Cluster Management for Kubernetes, applications are composed of multiple application resources. The foundational resources for Red Hat Advanced Cluster Management for Kubernetes applications are the **application** resource and the **deployable** resource.

In addition, you can use channel, subscription, and placement rule resources to help you deploy, update, and manage your overall applications.

Both single and multi-cluster applications use the same Kubernetes specifications, but multi-cluster applications involve more automation of the deployment and application management lifecycle.

All of the application component resources for Red Hat Advanced Cluster Management for Kubernetes applications are defined in YAML file spec sections. When you need to create or update an application component resource, you need to create or edit the appropriate spec section to include the labels for defining your resource.

View the following application resource samples:

- Channel samples
- Subscription samples
- Placement rule samples
- Application samples

1.1.6.5.1. Channel samples

View samples and YAML definitions that you can use to build your files. Channels (**channel.apps.open-cluster-management.io**) provide you with improved continuous integration and continuous delivery capabilities for creating and managing your Red Hat Advanced Cluster Management for Kubernetes applications. Learn more at Creating and managing channels.

1.1.6.5.1.1. Channel YAML structure

The following YAML structures show the required fields for a channel and some of the common optional fields. Your YAML structure needs to include some required fields and values. Depending on your application management requirements, you might need to include other optional fields and values. You can compose your own YAML content with any tool.

```
apiVersion: apps.open-cluster-management.io/v1
kind: Channel
metadata:
  name:
  namespace: # Each channel needs a unique namespace, except Git channel.
spec:
  sourceNamespaces:
  type:
  pathname:
  secretRef:
    name:
  gates:
    annotations:
  labels:
```

1.1.6.5.1.2. Channel YAML table

Field	Description
apiVersion	Required. Set the value to **apps.open-cluster-management.io/v1**.
kind	Required. Set the value to **Channel** to indicate that the resource is a channel.
metadata.name	Required. The name of the channel.
metadata.namespace	Required. The namespace for the channel; Each channel needs a unique namespace, except Git channel.
spec.sourceNamespaces	Optional. Identifies the namespace that the channel controller monitors for new or updated deployables to retrieve and promote to the channel.
spec.type	Required. The channel type. The supported types are: **HelmRepo**, **Git**, and **ObjectBucket**
spec.pathname	Required for **HelmRepo**, **Git**, **ObjectBucket** channels. For a **HelmRepo** channel, set the value to be the URL for the Helm repository. For an **ObjectBucket** channel, set the value to be the URL for the Object store. For a **Git** channel, set the value to be the HTTPS URL for the Git repository.
spec.secretRef.name	Optional. Identifies a Kubernetes Secret resource to use for authentication, such as for accessing a repository or chart. You can use a secret for authentication with only **HelmRepo**, **ObjectBucket**, and **Git** type channels.
spec.gates	Optional. Defines requirements for promoting a deployable within the channel. If no requirements are set, any deployable that is added to the channel namespace or source is promoted to the channel. **gates** do not apply for **HelmRepo** and **Git** channel types, only for **ObjectBucket** channel types.
spec.gates.annotations	Optional. The annotations for the channel. Deployables must have matching annotations to be included in the channel.
spec.labels	Optional. The labels for the channel.

The definition structure for a channel can resemble the following YAML content:

-

```
apiVersion: apps.open-cluster-management.io/v1
kind: Channel
metadata:
  name: predev-ch
  namespace: ns-ch
  labels:
    app: nginx-app-details
spec:
  type: HelmRepo
  pathname: https://kubernetes-charts.storage.googleapis.com/
```

```
  apiVersion: apps.open-cluster-management.io/v1
  kind: Channel
  metadata:
    name: qa
    namespace: ch-qa
  spec:
    sourceNamespaces:
    - default
    type: Namespace
    pathname: ch-qa
    gates:
      annotations:
        dev-ready: approved
```

```
apiVersion: apps.open-cluster-management.io/v1
kind: Deployable
metadata:
  labels:
    app: gbchn
    apps.open-cluster-management.io/channel: gbchn
    apps.open-cluster-management.io/channel-type: Namespace
    release: gbchn
  name: gbchn-service
  namespace: gbchn
spec:
  template:
    apiVersion: v1
    kind: Service
    metadata:
      labels:
        app: gbchn
        release: gbchn
      name: gbchn
    spec:
      ports:
      - port: 80
      selector:
        app: gbchn
```

1.1.6.5.1.3. Object store bucket (ObjectBucket) channel

The following example channel definition abstracts an object store bucket as a channel:

```
apiVersion: apps.open-cluster-management.io/v1
kind: Channel
metadata:
 name: dev
 namespace: ch-obj
spec:
 type: ObjectBucket
 pathname: [http://9.28.236.243:31311/dev] # URL is appended with the valid bucket name, which
matches the channel name.
 secretRef:
   name: miniosecret
 gates:
   annotations:
     dev-ready: true
```

1.1.6.5.1.4. Helm repository (**HelmRepo**) channel

The following example channel definition abstracts a Helm repository as a channel:

```
apiVersion: v1
kind: Namespace
metadata:
  name: hub-repo
---
apiVersion: apps.open-cluster-management.io/v1
kind: Channel
metadata:
  name: helm
  namespace: hub-repo
spec:
    pathname: [https://9.21.107.150:8443/helm-repo/charts] # URL points to a valid chart URL.
    configRef:
      name: insecure-skip-verify
    type: HelmRepo
---
apiVersion: v1
data:
  insecureSkipVerify: "true"
kind: ConfigMap
metadata:
  name: insecure-skip-verify
  namespace: hub-repo
```

The following channel definition shows another example of a Helm repository channel:

```
apiVersion: apps.open-cluster-management.io/v1
kind: Channel
metadata:
  name: predev-ch
  namespace: ns-ch
  labels:
    app: nginx-app-details
spec:
  type: HelmRepo
  pathname: https://kubernetes-charts.storage.googleapis.com/
```

-

1.1.6.5.1.5. Git (Git) repository channel

The following example channel definition shows an example of a channel for the Git Repository. In the following example, **secretRef** refers to the user identity used to access the Git repo that is specified in the **pathname**. If you have a public repo, you do not need the **secretRef**:

```
apiVersion: apps.open-cluster-management.io/v1
kind: Channel
metadata:
  name: hive-cluster-gitrepo
  namespace: gitops-cluster-lifecycle
spec:
  type: Git
  pathname: https://github.com/open-cluster-management/gitops-clusters.git
  secretRef:
    name: github-gitops-clusters
---
apiVersion: v1
kind: Secret
metadata:
  name: github-gitops-clusters
  namespace: gitops-cluster-lifecycle
data:
  user: dXNlcgo=            # Value of user and accessToken is Base 64 coded.
  accessToken: cGFzc3dvcmQ
```

1.1.6.5.1.6. Secret samples

Secrets (**Secret**) are Kubernetes resources that you can use to store authorization and other sensitive information, such as passwords, OAuth tokens, and SSH keys. By storing this information as secrets, you can separate the information from the application components that require the information to improve your data security. For more information about secrets, see Managing secrets.

The definition structure for a secret can resemble the following YAML content:

1.1.6.5.1.6.1. Secret YAML structure

```
apiVersion: v1
kind: Secret
metadata:
  annotations:
     apps.open-cluster-management.io/deployables: "true"
  name: [secret-name]
  namespace: [channel-namespace]
data:
  AccessKeyID: [ABCdeF1=] #Base64 encoded
  SecretAccessKey: [gHIjk2lmnoPQRST3uvw==] #Base64 encoded
```

1.1.6.5.2. Subscription samples

View samples and YAML definitions that you can use to build your files. As with channels, subscriptions (**subscription.apps.open-cluster-management.io**) provide you with improved continuous integration and continuous delivery capabilities for application management. Learn more at Creating and managing

subscriptions.

1.1.6.5.2.1. Subscription YAML structure

The following YAML structure shows the required fields for a subscription and some of the common optional fields. Your YAML structure needs to include certain required fields and values. Depending on your application management requirements, you might need to include other optional fields and values. You can compose your own YAML content with any tool:

```
apiVersion: apps.open-cluster-management.io/v1
kind: Subscription
metadata:
  name:
  namespace:
  labels:
spec:
  sourceNamespace:
  source:
  channel:
  name:
  packageFilter:
    version:
    labelSelector:
      matchLabels:
        package:
        component:
    annotations:
  packageOverrides:
  - packageName:
    packageAlias:
    - path:
      value:
  placement:
    local:
    clusters:
      name:
    clusterSelector:
    placementRef:
      name:
      kind: PlacementRule
  overrides:
    clusterName:
    clusterOverrides:
      path:
      value:
```

1.1.6.5.2.2. Subscription YAML table

Field	Description	
apiVersion	Required. Set the value to **apps.open-cluster-management.io/v1**.	

Field	Description	
kind	Required. Set the value to **Subscription** to indicate that the resource is a subscription.	
metadata.name	Required. The name for identifying the subscription.	
metadata.namespace	Required. The namespace resource to use for the subscription.	
metadata.labels	Optional. The labels for the subscription.	
spec.channel	Optional. The NamespaceName ("Namespace/Name") that defines the channel for the subscription. Define either the **channel**, or the **source**, or the **sourceNamespace** field. In general, use the **channel** field to point to the channel instead of using the **source** or **sourceNamespace** fields. If more than one field is defined, the first field that is defined is used.	
spec.sourceNamespace	Optional. The source namespace where deployables are stored on the Hub cluster. Use this field only for namespace channels. Define either the **channel**, or the **source**, or the **sourceNamespace** field. In general, use the **channel** field to point to the channel instead of using the **source** or **sourceNamespace** fields.	
spec.source	Optional. The path name ("URL") to the Helm repository where deployables are stored. Use this field for only Helm repository channels. Define either the **channel**, or the **source**, or the **sourceNamespace** field. In general, use the **channel** field to point to the channel instead of using the **source** or **sourceNamespace** fields.	

Field	Description	
spec.name	Required for **HelmRepo** type channels, but optional for **Namespace** and **ObjectBucket** type channels. The specific name for the target Helm chart or deployable within the channel. If neither the **name** or **packageFilter** are defined for channel types where the field is optional, all deployables are found and the latest version of each deployable is retrieved.	
spec.packageFilter	Optional. Defines the parameters to use to find target deployables or a subset of a deployables. If multiple filter conditions are defined, a deployable must meet all filter conditions.	
spec.packageFilter.version	Optional. The version or versions for the deployable. You can use a range of versions in the form **>1.0**, or **<3.0**. By default, the version with the most recent "creationTimestamp" value is used.	
spec.packageFilter.annotations	Optional. The annotations for the deployable.	
spec.packageOverrides	Optional. Section for defining overrides for the Kubernetes resource that is subscribed to by the subscription, such as a Helm chart, deployable, or other Kubernetes resource within a channel.	
spec.packageOverrides.packageName	Optional, but required for setting an override. Identifies the Kubernetes resource that is being overwritten.	
spec.packageOverrides.packageAlias	Optional. Gives an alias to the Kubernetes resource that is being overwritten.	

Field	Description	
spec.packageOverrides.packageOverrides	Optional. The configuration of parameters and replacement values to use to override the Kubernetes resource. For more information, see Configuring package overrides.	
spec.placement	Required. Identifies the subscribing clusters where deployables need to be placed, or the placement rule that defines the clusters. Use the placement configuration to define values for multi-cluster deployments.	
spec.local	Optional, but required for a stand-alone cluster or cluster that you want to manage directly. Defines whether the subscription must be deployed locally. Set the value to **true** to have the subscription synchronize with the specified channel. Set the value to **false** to prevent the subscription from subscribing to any resources from the specified channel. Use this field when your cluster is a stand-alone cluster or you are managing this cluster directly. If your cluster is part of a multi-cluster and you do not want to manage the cluster directly, use only one of **clusters**, **clusterSelector**, or **placementRef** to define where your subscription is to be placed. If your cluster is the Hub of a multi-cluster and you want to manage the cluster directly, you must register the Hub as a managed cluster before the subscription operator can subscribe to resources locally.	

Field	Description	
spec.placement.clusters	Optional. Defines the clusters where the subscription is to be placed. Use only one of **clusters**, **clusterSelector**, or **placementRef** to define where your subscription is to be placed for a multi-cluster. If your cluster is a stand-alone cluster that is not your Hub cluster, you can also use **local**.	
spec.placement.clusters.name	Optional, but required for defining the subscribing clusters. The name or names of the subscribing clusters.	
spec.placement.clusterSelector	Optional. Defines the label selector to use to identify the clusters where the subscription is to be placed. Use only one of **clusters**, **clusterSelector**, or **placementRef** to define where your subscription is to be placed for a multi-cluster. If your cluster is a stand-alone cluster that is not your Hub cluster, you can also use **local**.	
spec.placement.placementRef	Optional. Defines the placement rule to use for the subscription. Use only one of **clusters**, **clusterSelector** , or **placementRef** to define where your subscription is to be placed for a multi-cluster. If your cluster is a stand-alone cluster that is not your Hub cluster, you can also use **local**.	
spec.placement.placementRef.name	Optional, but required for using a placement rule. The name of the placement rule for the subscription.	
spec.placement.placementRef.kind	Optional, but required for using a placement rule. Set the value to **PlacementRule** to indicate that a placement rule is used for deployments with the subscription.	

Field	Description	
spec.overrides	Optional. Any parameters and values that need to be overridden, such as cluster-specific settings.	
spec.overrides.clusterName	Optional. The name of the cluster or clusters where parameters and values are being overridden.	
spec.overrides.clusterOverrides	Optional. The configuration of parameters and values to override.	
spec.timeWindow	Optional. Defines the settings for configuring a time window when the subscription is active or blocked.	
spec.timeWindow.type	Optional, but required for configuring a time window. Indicates whether the subscription is active or blocked during the configured time window. Deployments for the subscription occur only when the subscription is active.	
spec.timeWindow.location	Optional, but required for configuring a time window. The time zone of the configured time range for the time window. All time zones must use the Time Zone (tz) database name format. For more information, see Time Zone Database.	
spec.timeWindow.daysofweek	Optional, but required for configuring a time window. Indicates the days of the week when the time range is applied to create a time window. The list of days must be defined as an array, such as **daysofweek: ["Monday", "Wednesday", "Friday"]**.	

Field	Description	
spec.timeWindow.hours	Optional, but required for configuring a time window. Defined the time range for the time window. A start time and end time for the hour range must be defined for each time window. You can define multiple time window ranges for a subscription.	
spec.timeWindow.hours.start	Optional, but required for configuring a time window. The timestamp that defines the beginning of the time window. The timestamp must use the Go programming language Kitchen format **"hh:mmpm"**. For more information, see Constants.	
spec.timeWindow.hours.end	Optional, but required for configuring a time window. The timestamp that defines the ending of the time window. The timestamp must use the Go programming language Kitchen format **"hh:mmpm"**. For more information, see Constants.	-→

Notes:

- When you are defining your YAML, a subscription can use **packageFilters** to point to multiple Helm charts, deployables, or other Kubernetes resources. The subscription, however, only deploys the latest version of one chart, or deployable, or other resource.
- Annotations are used by a subscription operator for **Namespace** type channels to search for versions of a deployable. The subscription operator searches the versions to find the appropriate deployable version to retrieve. If your channel is a **Namespace** channel, include the annotations for identifying the deployable version.
- For time windows, when you are defining the time range for a window, the start time must be set to occur before the end time. If you are defining multiple time windows for a subscription, the time ranges for the windows cannot overlap. The actual time ranges are based on the **subscription-controller** container time, which can be set to a different time and location than the time and location that you are working within.
- Within your subscription spec, you can also define the placement of a Helm release or deployable as part of the subscription definition. Similar to the definition for deployables, each subscription can reference an existing placement rule, or define a placement rule directly within the subscription definition.
- When you are defining where to place your subscription in the **spec.placement** section, use only one of **clusters**, **clusterSelector**, or **placementRef** for a multi-cluster environment. If you include more than one of **clusters**, **clusterSelector**, or **placementRef**, the following priority is

used to determine which setting the subscription operator uses:

a. **placementRef**

b. **clusters**

c. **clusterSelector**

Your subscription can resemble the following YAML content:

```
apiVersion: apps.open-cluster-management.io/v1
kind: Subscription
metadata:
  name: nginx
  namespace: ns-sub-1
  labels:
    app: nginx-app-details
spec:
  channel: ns-ch/predev-ch
  name: nginx-ingress
  packageFilter:
    version: "1.36.x"
  placement: # Placement rules help you facilitate multi-cluster deployments, see placement rules documentation.
    placementRef:
      kind: PlacementRule
      name: towhichcluster
  overrides: # See Deployable documentation for more about overrides. Include overrides for any single cluster than requires some different settings
  - clusterName: "/"
    clusterOverrides:
    - path: "metadata.namespace"
      value: default
```

1.1.6.5.2.3. Subscription file samples

```
apiVersion: apps.open-cluster-management.io/v1
kind: Subscription
metadata:
  name: nginx
  namespace: ns-sub-1
  labels:
    app: nginx-app-details
spec:
  channel: ns-ch/predev-ch
  name: nginx-ingress
```

1.1.6.5.2.3.1. Subscription time window example

The following example subscription includes multiple configured time windows. A time window occurs between 10:20 AM and 10:30 AM occurs every Monday, Wednesday, and Friday. A time window also occurs between 12:40 PM and 1:40 PM every Monday, Wednesday, and Friday. The subscription is active only during these six weekly time windows for deployments to begin.

```
apiVersion: apps.open-cluster-management.io/v1
kind: Subscription
metadata:
  name: nginx
  namespace: ns-sub-1
  labels:
    app: nginx-app-details
spec:
  channel: ns-ch/predev-ch
  name: nginx-ingress
  packageFilter:
    version: "1.36.x"
  placement:
    placementRef:
      kind: PlacementRule
      name: towhichcluster
  timewindow:
    windowtype: "active" #Enter active or blocked depending on the purpose of the type.
    location: "America/Los_Angeles"
    daysofweek: ["Monday", "Wednesday", "Friday"]
    hours:
      - start: "10:20AM"
        end: "10:30AM"
      - start: "12:40PM"
        end: "1:40PM"
```

1.1.6.5.2.3.2. Subscription with overrides example

The following example includes package overrides to define a different release name of the Helm release for Helm chart. A package override setting is used to set the name **my-nginx-ingress-releaseName** as the different release name for the **nginx-ingress** Helm release.

```
apiVersion: apps.open-cluster-management.io/v1
kind: Subscription
metadata:
  name: simple
  namespace: default
spec:
  channel: ns-ch/predev-ch
  name: nginx-ingress
  packageOverrides:
  - packageName: nginx-ingress
    packageAlias: my-nginx-ingress-releaseName
    packageOverrides:
    - path: spec
      value:
        defaultBackend:
          replicaCount: 3
  placement:
    local: false
```

1.1.6.5.2.3.3. Helm repository subscription example

The following subscription automatically pulls the latest **nginx** Helm release for the version **1.36.x**. The Helm release deployable is placed on the **my-development-cluster-1** cluster when a new version is available in the source Helm repository.

The **spec.packageOverrides** section shows optional parameters for overriding values for the Helm release. The override values are merged into the Helm release **values.yaml** file, which is used to retrieve the configurable variables for the Helm release.

```
apiVersion: apps.open-cluster-management.io/v1
kind: Subscription
metadata:
  name: nginx
  namespace: ns-sub-1
  labels:
    app: nginx-app-details
spec:
  channel: ns-ch/predev-ch
  name: nginx-ingress
  packageFilter:
    version: "1.36.x"
  placement:
    clusters:
    - name: my-development-cluster-1
  packageOverrides:
  - packageName: my-server-integration-prod
    packageOverrides:
    - path: spec
      value:
        persistence:
          enabled: false
          useDynamicProvisioning: false
        license: accept
        tls:
          hostname: my-mcm-cluster.icp
        sso:
          registrationImage:
            pullSecret: hub-repo-docker-secret
```

1.1.6.5.2.3.4. Git repository subscription example

1.1.6.5.2.3.4.1. Subscribing specific branch and directory of Git repository

```
  apiVersion: apps.open-cluster-management.io/v1
  kind: Subscription
  metadata:
    name: sample-subscription
    namespace: default
    annotations:
      apps.open-cluster-management.io/git-path: sample_app_1/dir1
      apps.open-cluster-management.io/git-branch: branch1
  spec:
    channel: default/sample-channel
    placement:
```

```
    placementRef:
      kind: PlacementRule
      name: dev-clusters
```

In this example subscription, the annotation **apps.open-cluster-management.io/git-path** indicates that the subscription subscribes to all Helm charts and Kubernetes resources within the **sample_app_1/dir1** directory of the Git repository that is specified in the channel. The subscription subscribes to **master** branch by default. In this example subscription, the annotation **apps.open-cluster-management.io/git-branch: branch1** is specified to subscribe to **branch1** branch of the repository.

1.1.6.5.2.3.4.2. Adding a .kubernetesignore file

You can include a **.kubernetesignore** file within your Git repository root directory, or within the **apps.open-cluster-management.io/git-path** directory that is specified in subscription's annotations.

You can use this **.kubernetesignore** file to specify patterns of files or subdirectories, or both, to ignore when the subscription deploys Kubernetes resources or Helm charts from the repository.

You can also use the **.kubernetesignore** file for fine-grain filtering to selectively apply Kubernetes resources. The pattern format of the **.kubernetesignore** file is the same as a **.gitignore** file.

If the **apps.open-cluster-management.io/git-path** annotation is not defined, the subscription looks for a **.kubernetesignore** file in the repository root directory. If the **apps.open-cluster-management.io/git-path** field is defined, the subscription looks for the **.kubernetesignore** file in the **apps.open-cluster-management.io/git-path** directory. Subscriptions do not search in any other directory for a **.kubernetesignore** file.

1.1.6.5.2.3.4.3. Applying Kustomize

If there is **kustomization.yaml** or **kustomization.yml** file in a subscribed Git folder, kustomize is applied.

You can use **spec.packageOverrides** to override **kustomization** at the subscription deployment time.

```
apiVersion: apps.open-cluster-management.io/v1
kind: Subscription
metadata:
  name: example-subscription
  namespace: default
spec:
  channel: some/channel
  packageOverrides:
  - packageName: kustomization
    packageOverrides:
    - value: |
patchesStrategicMerge:
- patch.yaml
```

In order to override **kustomization.yaml** file, **packageName: kustomization** is required in **packageOverrides**. The override either adds new entries or updates existing entries. It does not remove existing entries.

1.1.6.5.2.3.4.4. Enabling Git WebHook

By default, Git channel subscriptions clone the Git repository specified in the channel once every

By default, a Git channel subscription clones the Git repository specified in the channel every minute and applies changes when the commit ID has changed. Alternatively, you can configure your subscription to apply changes only when the Git repository sends repo PUSH and PULL webhook event notifications.

In order to configure webhook in a Git repository, you need a target webhook payload URL and optionally a secret.

1.1.6.5.2.3.4.4.1. Payload URL

Create a route (ingress) in the hub cluster to expose the subscription operator's webhook event listener service.

```
  oc create route passthrough --service=multicluster-operators-subscription -n open-cluster-
management
```

Then, use **oc get route multicluster-operators-subscription -n open-cluster-management** command to find the externally-reachable hostname. The webhook payload URL:

```
https://<externally-reachable hostname>/webhook`
```

1.1.6.5.2.3.4.4.2. WebHook secret

WebHook secret is optional. Create a Kubernetes secret in the channel namespace. The secret must contain **data.secret**. See the following example:

```
apiVersion: v1
kind: Secret
metadata:
  name: my-github-webhook-secret
data:
  secret: BASE64_ENCODED_SECRET
```

The value of **data.secret** is the base-64 encoded WebHook secret you are going to use.

Best practice: Use a unique secret for each Git repository.

1.1.6.5.2.3.4.4.3. Configuring WebHook in Git repository

Use the payload URL and webhook secret to configure WebHook in your Git repository.

1.1.6.5.2.3.4.4.4. Enable WebHook event notification in channel

Annotate the subscription channel. See the following example:

```
oc annotate channel.apps.open-cluster-management.io <channel name> apps.open-cluster-
management.io/webhook-enabled="true"
```

If you used a secret to configure WebHook, annotate the channel with this as well where **<the_secret_name>** is the kubernetes secret name containing webhook secret.

```
oc annotate channel.apps.open-cluster-management.io <channel name> apps.open-cluster-
management.io/webhook-secret="<the_secret_name>"
```

1.1.6.5.2.3.4.4.5. Subscriptions of webhook-enabled channel

No webhook specific configuration is needed in subscriptions.

1.1.6.5.3. Placement rule samples

Placement rules (**placementrule.apps.open-cluster-management.io**) define the target clusters where deployables can be deployed. Use placement rules to help you facilitate the multi-cluster deployment of your deployables.

1.1.6.5.3.1. Placement rule YAML structure

The following YAML structure shows the required fields for a placement rule and some of the common optional fields. Your YAML structure needs to include some required fields and values. Depending on your application management requirements, you might need to include other optional fields and values. You can compose your own YAML content with any tool.

```
apiVersion: apps.open-cluster-management.io/v1
kind: PlacementRule
  name:
  namespace:
  resourceVersion:
  labels:
    app:
    chart:
    release:
    heritage:
  selfLink:
  uid:
spec:
  clusterSelector:
    matchLabels:
      datacenter:
      environment:
  clusterReplicas:
  clusterConditions:
  ResourceHint:
    type:
    order:
  Policies:
```

1.1.6.5.3.2. Placement rule YAML values table

Field	Description
apiVersion	Required. Set the value to **apps.open-cluster-management.io/v1**.
kind	Required. Set the value to **PlacementRule** to indicate that the resource is a placement rule.
metadata.name	Required. The name for identifying the placement rule.

Field	Description
metadata.namespace	Required. The namespace resource to use for the placement rule.
metadata.resourceVersion	Optional. The version of the placement rule resource.
metadata.labels	Optional. The labels for the placement rule.
spec.clusterSelector	Optional. The labels for identifying the target clusters
spec.clusterSelector.matchLabels	Optional. The labels that must exist for the target clusters.
status.decisions	Optional. Defines the target clusters where deployables are placed.
status.decisions.clusterName	Optional. The name of a target cluster
status.decisions.clusterNamespace	Optional. The namespace for a target cluster.
spec.clusterReplicas	Optional. The number of replicas to create.
spec.clusterConditions	Optional. Define any conditions for the cluster.
spec.ResourceHint	Optional. If more than one cluster matches the labels and values that you provided in the previous fields, you can specify a resource specific criteria to select the clusters. For example, you can select the cluster with the most available CPU cores.
spec.ResourceHint.type	Optional. Set the value to either **cpu** to select clusters based on available CPU cores or **memory** to select clusters based on available memory resources.
spec.ResourceHint.order	Optional. Set the value to either **asc** for ascending order, or **desc** for descending order.
spec.Policies	Optional. The policy filters for the placement rule.

1.1.6.5.3.3. Placement rule sample files

Existing placement rules can include the following fields that indicate the status for the placement rule. This status section is appended after the **spec** section in the YAML structure for a rule.

```
status:
  decisions:
```

```
    clusterName:
    clusterNamespace:
```

Field	Description
status	The status information for the placement rule.
status.decisions	Defines the target clusters where deployables are placed.
status.decisions.clusterName	The name of a target cluster
status.decisions.clusterNamespace	The namespace for a target cluster.

- Example 1

```
apiVersion: apps.open-cluster-management.io/v1
kind: PlacementRule
metadata:
  name: gbapp-gbapp
  namespace: development
  labels:
    app: gbapp
spec:
  clusterSelector:
    matchLabels:
      environment: Dev
  clusterReplicas: 1
status:
  decisions:
    - clusterName: local-cluster
      clusterNamespace: local-cluster
```

- Example 2

```
apiVersion: apps.open-cluster-management.io/v1
kind: PlacementRule
metadata:
  name: towhichcluster
  namespace: ns-sub-1
  labels:
    app: nginx-app-details
spec:
  clusterReplicas: 1
  clusterConditions:
    - type: ManagedClusterConditionAvailable
      status: "True"
  clusterSelector:
    matchExpressions:
    - key: environment
```

```
    operator: In
    values:
    - dev
```

1.1.6.5.4. Application samples

View samples and YAML definitions that you can use to build your files. Applications (**Application.app.k8s.io**) in Red Hat Advanced Cluster Management for Kubernetes are used for viewing the application components.

For more information about creating and managing applications, see Creating and managing application resources.

1.1.6.5.4.1. Application YAML structure

To compose the application definition YAML content for creating or updating an application resource, your YAML structure needs to include some required fields and values. Depending on your application requirements or application management requirements, you might need to include other optional fields and values.

The following YAML structure shows the required fields for an application and some of the common optional fields.

```
apiVersion: app.k8s.io/v1beta1
kind: Application
metadata:
  name:
  namespace:
  resourceVersion:
  annotations:
  labels:
    app:
    chart:
    heritage:
    name:
    release:
spec:
  componentKinds:
  - group:
    kind:
  descriptor:
  selector:
    matchExpressions:
    - key:
      operator:
      values:
```

1.1.6.5.4.2. Application YAML table

Field	Description
apiVersion	Required. Set the value to **app.k8s.io/v1beta1**.

Field	Description
kind	Required. Set the value to **Application** to indicate the resource is an application resource.
metadata.name	Required. The name for identifying the application resource.
metadata.namespace	The namespace resource to use for the application.
metadata.resourceVersion	The version of the application resource.
metadata.annotations	Optional. The annotations for the application.
metadata.labels	Optional. The labels for the deployable.
spec.componentKinds	Optional. The list of the kinds of resources to be associated with the application.
spec.selector.matchExpressions	Optional. Label selectors for associating other Kubernetes resources with the application.
spec.selector.matchExpressions.key	Required when defining label selectors. The Kubernetes label key that a resource needs to match.
spec.selector.matchExpressions.values	Required when defining label selectors. The Kubernetes label values that a resource needs to match.

The spec for defining these applications is based on the Application metadata descriptor custom resource definition that is provided by the Kubernetes Special Interest Group (SIG). You can use this definition to help you compose your own application YAML content. For more information about this definition, see Kubernetes SIG Application CRD community specification .

1.1.6.5.4.3. Application file samples

The definition structure for an application can resemble the following example YAML content:

```
apiVersion: app.k8s.io/v1beta1
kind: Application
metadata:
  labels:
    app: nginx-app-details
  name: nginx-app-3
  namespace: ns-sub-1
spec:
  componentKinds:
  - group: apps.open-cluster-management.io
    kind: Subscription
  selector:
```

```
    matchLabels:
      app: nginx-app-details
status: {}
```