Red Hat Advanced Cluster Management for Kubernetes 2.12

multicluster engine operator with Red Hat Advanced Cluster Management

multicluster engine operator with Red Hat Advanced Cluster Management integration

Last Updated: 2024-11-05

Red Hat Advanced Cluster Management for Kubernetes 2.12 multicluster engine operator with Red Hat Advanced Cluster Management

multicluster engine operator with Red Hat Advanced Cluster Management integration

Legal Notice

Abstract

If you are using multicluster engine operator and then you install Red Hat Advanced Cluster Management, you can access more multicluster management features.

Table of Contents

CHAPTER 1. MULTICLUSTER ENGINE OPERATOR WITH RED HAT ADVANCED CLUSTER MANAGEMENT INTEGRATION

If you are using multicluster engine operator and then you install Red Hat Advanced Cluster Management, you can access more multicluster management features, such as *Observability* and *Policy*.

For integrated capability, see the following requirements:

- You need to install Red Hat Advanced Cluster Management. See the Red Hat Advanced Cluster Management Installing and upgrading documentation.
- See MultiClusterHub advanced configuration for details about Red Hat Advanced Cluster Management after you install.

See the following procedures for multicluster engine operator and Red Hat Advanced Cluster Management multicluster management:

- Discovering multicluster engine operator hosted clusters in Red Hat Advanced Cluster Management
- Automating import for discovered hosted clusters
- Automating import for discovered OpenShift Service on AWS clusters
- Observability integration
- SiteConfig operator

1.1. DISCOVERING MULTICLUSTER ENGINE OPERATOR HOSTED CLUSTERS IN RED HAT ADVANCED CLUSTER MANAGEMENT

If you have multicluster engine operator clusters that are hosting multiple *hosted clusters*, you can bring those hosted clusters to a Red Hat Advanced Cluster Management hub cluster to manage with Red Hat Advanced Cluster Management management components, such as *Application lifecycle* and *Governance*.

Those hosted clusters can be automatically discovered and imported as managed clusters.

Note: Since the hosted control planes run on the managed multicluster engine operator cluster nodes, the number of hosted control planes that the cluster can host is determined by the resource availability of managed multicluster engine operator cluster nodes, as well as the number of managed multicluster engine operator clusters. You can add more nodes or managed clusters to host more hosted control planes.

Required access: Cluster administrator

- Prerequisites
- Configuring Red Hat Advanced Cluster Management to import multicluster engine operator clusters
- Importing multicluster engine operator manually
- Discovering hosted clusters from multicluster engine operator

1.1.1. Prerequisites

- You need one or more multicluster engine operator clusters.
- You need a Red Hat Advanced Cluster Management cluster that is set as your hub cluster.
- Install the **clusteradm** CLI by running the following command:

```
curl -L https://raw.githubusercontent.com/open-cluster-management-
io/clusteradm/main/install.sh | bash
```

1.1.2. Configuring Red Hat Advanced Cluster Management to import multicluster engine operator clusters

multicluster engine operator has a **local-cluster**, which is a hub cluster that is managed. The following default addons are enabled for this **local-cluster** in the **open-cluster-management-agent-addon** namespace:

- **cluster-proxy**
- **managed-serviceaccount**
- **work-manager**

1.1.2.1. Configuring add-ons

When your multicluster engine operator is imported into Red Hat Advanced Cluster Management, Red Hat Advanced Cluster Management enables the same set of add-ons to manage the multicluster engine operator.

Install those add-ons in a different multicluster engine operator namespace so that the multicluster engine operator can self-manage with the **local-cluster** add-ons while Red Hat Advanced Cluster Management manages multicluster engine operator at the same time. Complete the following procedure:

1. Log in to your Red Hat Advanced Cluster Management with the CLI.
2. Create the **addonDeploymentConfig** resource to specify a different add-on installation namespace. See the following example where **agentInstallNamespace** points to **open-cluster-management-agent-addon-discovery**:

```
apiVersion: addon.open-cluster-management.io/v1alpha1
kind: addonDeploymentConfig
metadata:
  name: addon-ns-config
  namespace: multicluster-engine
spec:
  agentInstallNamespace: open-cluster-management-agent-addon-discovery
```

3. Run **oc apply -f <filename>.yaml** to apply the file.
4. Update the existing **ClusterManagementAddOn** resources for the add-ons so that the add-ons are installed in the **open-cluster-management-agent-addon-discovery** namespace that is specified in the **addonDeploymentConfig** resource that you created. See the following example with **open-cluster-management-global-set** as the namespace:

```
apiVersion: addon.open-cluster-management.io/v1alpha1
kind: ClusterManagementAddOn
metadata:
  name: work-manager
spec:
  addonMeta:
    displayName: work-manager
  installStrategy:
    placements:
    - name: global
      namespace: open-cluster-management-global-set
      rolloutStrategy:
        type: All
    type: Placements
```

a. Add the **addonDeploymentConfigs** to the **ClusterManagementAddOn**. See the following example:

```
apiVersion: addon.open-cluster-management.io/v1alpha1
kind: ClusterManagementAddOn
metadata:
  name: work-manager
spec:
  addonMeta:
    displayName: work-manager
  installStrategy:
    placements:
    - name: global
      namespace: open-cluster-management-global-set
      rolloutStrategy:
        type: All
      configs:
      - group: addon.open-cluster-management.io
        name: addon-ns-config
        namespace: multicluster-engine
        resource: addondeploymentconfigs
    type: Placements
```

b. Add the **addonDeploymentConfig** to the **managed-serviceaccount**. See the following example:

```
apiVersion: addon.open-cluster-management.io/v1alpha1
kind: ClusterManagementAddOn
metadata:
  name: managed-serviceaccount
spec:
  addonMeta:
    displayName: managed-serviceaccount
  installStrategy:
    placements:
    - name: global
      namespace: open-cluster-management-global-set
      rolloutStrategy:
        type: All
      configs:
```

```
      - group: addon.open-cluster-management.io
        name: addon-ns-config
        namespace: multicluster-engine
        resource: addondeploymentconfigs
    type: Placements
```

c. Add the **addondeploymentconfigs** value to the **ClusterManagementAddOn** resource named, **cluster-proxy**. See the following example:

```
apiVersion: addon.open-cluster-management.io/v1alpha1
kind: ClusterManagementAddOn
metadata:
  name: cluster-proxy
spec:
  addonMeta:
    displayName: cluster-proxy
  installStrategy:
    placements:
    - name: global
      namespace: open-cluster-management-global-set
      rolloutStrategy:
        type: All
      configs:
      - group: addon.open-cluster-management.io
        name: addon-ns-config
        namespace: multicluster-engine
        resource: addondeploymentconfigs
    type: Placements
```

5. Run the following command to verify that the add-ons for the Red Hat Advanced Cluster Management **local-cluster** are re-installed into the namespace that you specified:

```
oc get deployment -n open-cluster-management-agent-addon-discovery
```

See the following output example:

```
NAME                                READY   UP-TO-DATE   AVAILABLE    AGE
cluster-proxy-proxy-agent           1/1     1            1            24h
klusterlet-addon-workmgr            1/1     1            1            24h
managed-serviceaccount-addon-agent  1/1     1            1            24h
```

1.1.2.2. Creating a *KlusterletConfig* resource

multicluster engine operator has a local-cluster, which is a hub cluster that is managed. A resource named **klusterlet** is created for this local-cluster.

When your multicluster engine operator is imported into Red Hat Advanced Cluster Management, Red Hat Advanced Cluster Management installs the klusterlet with the same name, **klusterlet**, to manage the multicluster engine operator. This conflicts with the multicluster engine operator local-cluster klusterlet.

You need to create a **KlusterletConfig** resource that is used by **ManagedCluster** resources to import multicluster engine operator clusters so that the klusterlet is installed with a different name to avoid the conflict. Complete the following procedure:

1. Create a **KlusterletConfig** resource using the following example. When this **KlusterletConfig** resource is referenced in a managed cluster, the value in the **spec.installMode.noOperator.postfix** field is used as a suffix to the klusterlet name, such as **klusterlet-mce-import**:

```
kind: KlusterletConfig
apiVersion: config.open-cluster-management.io/v1alpha1
metadata:
  name: mce-import-klusterlet-config
spec:
  installMode:
    type: noOperator
    noOperator:
      postfix: mce-import
```

2. Run **oc apply -f <filename>.yaml** to apply the file.

1.1.2.3. Configure for backup and restore

Since you installed Red Hat Advanced Cluster Management, you can also use the *Backup and restore* feature.

If the hub cluster is restored in a disaster recovery scenario, the imported multicluster engine operator clusters and hosted clusters are imported to the newer Red Hat Advanced Cluster Management hub cluster.

In this scenario, you need to restore the previous configurations as part of Red Hat Advanced Cluster Management hub cluster restore.

Add the **backup=true** label to enable backup. See the following steps for each add-on:

- For your **addon-ns-config**, run the following command:

```
oc label addondeploymentconfig addon-ns-config -n multicluster-engine cluster.open-cluster-management.io/backup=true
```

- For your **hypershift-addon-deploy-config**, run the following command:

```
oc label addondeploymentconfig hypershift-addon-deploy-config -n multicluster-engine cluster.open-cluster-management.io/backup=true
```

- For your **work-manager**, run the following command:

```
oc label clustermanagementaddon work-manager cluster.open-cluster-management.io/backup=true
```

- For your `cluster-proxy `, run the following command:

```
oc label clustermanagementaddon cluster-proxy cluster.open-cluster-management.io/backup=true
```

- For your **managed-serviceaccount**, run the following command:

```
oc label clustermanagementaddon managed-serviceaccount cluster.open-cluster-
management.io/backup=true
```

- For your **mce-import-klusterlet-config**, run the following command:

```
oc label KlusterletConfig mce-import-klusterlet-config cluster.open-cluster-
management.io/backup=true
```

1.1.3. Importing multicluster engine operator manually

To manually import an multicluster engine operator cluster from your Red Hat Advanced Cluster Management cluster, complete the following procedure:

1. From your Red Hat Advanced Cluster Management cluster, create a **ManagedCluster** resource manually to import an multicluster engine operator cluster. See the following file example:

```
apiVersion: cluster.open-cluster-management.io/v1
kind: ManagedCluster
metadata:
  annotations:
    agent.open-cluster-management.io/klusterlet-config: mce-import-klusterlet-config 1
  name: mce-a 2
spec:
  hubAcceptsClient: true
  leaseDurationSeconds: 60
```

1 The **mce-import-klusterlet-config** annotation references the **KlusterletConfig** resource that you created in the previous step to install the Red Hat Advanced Cluster Management klusterlet with a different name in multicluster engine operator.

2 The example imports an multicluster engine operator managed cluster named **mce-a**.

2. Run **oc apply -f <filename>.yaml** to apply the file.

3. Create the **auto-import-secret** secret that references the **kubeconfig** of the multicluster engine operator cluster. Go to *Importing a cluster by using the auto import secret* in Importing a managed cluster by using the CLI to add the auto import secret to complete the multicluster engine operator auto-import process.
After you create the auto import secret in the multicluster engine operator managed cluster namespace in the Red Hat Advanced Cluster Management cluster, the managed cluster is registered.

4. Run the following command to get the status:

```
oc get managedcluster
```

See following example output with the status and example URLs of managed clusters:

```
NAME          HUB ACCEPTED   MANAGED CLUSTER URLS           JOINED   AVAILABLE
AGE
local-cluster  true          https://<api.acm-hub.com:port>  True     True      44h
mce-a          true          https://<api.mce-a.com:port>    True     True      27s
```

Important: Do not enable any other Red Hat Advanced Cluster Management add-ons for the imported multicluster engine operator.

1.1.4. Discovering hosted clusters

After all your multicluster engine operator clusters are imported into Red Hat Advanced Cluster Management, you need to enable the **hypershift-addon** for those managed multicluster engine operator clusters to discover the hosted clusters.

Default add-ons are installed into a different namespace in the previous procedures. Similarly, you install the **hypershift-addon** into a different namespace in multicluster engine operator so that the add-ons agent for multicluster engine operator local-cluster and the agent for Red Hat Advanced Cluster Management can work in multicluster engine operator.

Important: For all the following commands, replace **<managed-cluster-names>** with comma-separated managed cluster names for multicluster engine operator.

1. Run the following command to set the **agentInstallNamespace** namespace of the add-on to **open-cluster-management-agent-addon-discovery**:

```
oc patch addondeploymentconfig hypershift-addon-deploy-config -n multicluster-engine --type=merge -p '{"spec":{"agentInstallNamespace":"open-cluster-management-agent-addon-discovery"}}'
```

2. Run the following command to disable metrics and to disable the HyperShift operator management:

```
oc patch addondeploymentconfig hypershift-addon-deploy-config -n multicluster-engine --type=merge -p '{"spec":{"customizedVariables":[{"name":"disableMetrics","value": "true"},{"name":"disableHOManagement","value": "true"}]}}'
```

3. Run the following command to enable the **hypershift-addon** for multicluster engine operator:

```
clusteradm addon enable --names hypershift-addon --clusters <managed-cluster-names>
```

4. You can get the multicluster engine operator managed cluster names by running the following command in Red Hat Advanced Cluster Management.

```
oc get managedcluster
```

5. Log into multicluster engine operator clusters and verify that the **hypershift-addon** is installed in the namespace that you specified. Run the following command:

```
oc get deployment -n open-cluster-management-agent-addon-discovery
```

See the following example output that lists the add-ons:

```
NAME                               READY   UP-TO-DATE   AVAILABLE   AGE
cluster-proxy-proxy-agent          1/1     1            1           24h
klusterlet-addon-workmgr           1/1     1            1           24h
hypershift-addon-agent             1/1     1            1           24h
managed-serviceaccount-addon-agent 1/1     1            1           24h
```

Red Hat Advanced Cluster Management deploys the **hypershift-addon**, which is the discovery agent that discovers hosted clusters from multicluster engine operator. The agent creates the corresponding **DiscoveredCluster** custom resource in the multicluster engine operator managed cluster namespace in the Red Hat Advanced Cluster Management hub cluster when the hosted cluster **kube-apiserver** becomes available.

You can view your discovered clusters in the console.

1. Log into hub cluster console and navigate to **All Clusters** > **Infrastructure** > **Clusters**.

2. Find the *Discovered clusters* tab to view all discovered hosted clusters from multicluster engine operator with type **MultiClusterEngineHCP**.

Next, visit Automating import for discovered hosted clusters to learn how to automatically import clusters.

1.2. AUTOMATING IMPORT FOR DISCOVERED HOSTED CLUSTERS

Automate the import of hosted clusters by using the **DiscoveredCluster** resource for faster cluster management, without manually importing individual clusters.

When you automatically import a discovered hosted cluster into Red Hat Advanced Cluster Management, all Red Hat Advanced Cluster Management add-ons are enabled so that you can start managing the hosted clusters with the available management tools.

The hosted cluster is also *auto-imported* into multicluster engine operator. Through the multicluster engine operator console, you can manage the hosted cluster lifecycle. However, you cannot manage the hosted cluster lifecycle from the Red Hat Advanced Cluster Management console.

Required access: Cluster administrator

- Prerequisites
- Configure settings for automatic import
- Creating the placement definition
- Binding the import policy to a placement definition

1.2.1. Prerequisites

- You need Red Hat Advanced Cluster Management installed. See the Red Hat Advanced Cluster Management Installing and upgrading documentation.

- You need to learn about *Policies*. See the introduction to Governance in the Red Hat Advanced Cluster Management documentation.

1.2.2. Configuring settings for automatic import

Discovered hosted clusters from managed multicluster engine operator clusters are represented in **DiscoveredCluster** custom resources, which are located in the managed multicluster engine operator cluster namespace in Red Hat Advanced Cluster Management. See the following **DiscoveredCluster** resource and namespace example:

```
apiVersion: discovery.open-cluster-management.io/v1
```

```
kind: DiscoveredCluster
metadata:
  creationTimestamp: "2024-05-30T23:05:39Z"
  generation: 1
  labels:
    hypershift.open-cluster-management.io/hc-name: hosted-cluster-1
    hypershift.open-cluster-management.io/hc-namespace: clusters
  name: hosted-cluster-1
  namespace: mce-1
  resourceVersion: "1740725"
  uid: b4c36dca-a0c4-49f9-9673-f561e601d837
spec:
  apiUrl: https://a43e6fe6dcef244f8b72c30426fb6ae3-ea3fec7b113c88da.elb.us-west-
1.amazonaws.com:6443
  cloudProvider: aws
  creationTimestamp: "2024-05-30T23:02:45Z"
  credential: {}
  displayName: mce-1-hosted-cluster-1
  importAsManagedCluster: false
  isManagedCluster: false
  name: hosted-cluster-1
  openshiftVersion: 0.0.0
  status: Active
  type: MultiClusterEngineHCP
```

These discovered hosted clusters are not automatically imported into Red Hat Advanced Cluster Management until the **spec.importAsManagedCluster** field is set to **true**. Learn how to use a Red Hat Advanced Cluster Management policy to automatically set this field to **true** for all **type.MultiClusterEngineHCP** within **DiscoveredCluster** resources so that discovered hosted clusters are immediately automatically imported into Red Hat Advanced Cluster Management.

Configure your Policy to import all your discovered hosted clusters automatically. Log in to your hub cluster from the CLI to complete the following procedure:

1. Create a YAML file for your **DiscoveredCluster** custom resource and edit the configuration that is referenced in the following example:

```
apiVersion: policy.open-cluster-management.io/v1
kind: Policy
metadata:
  name: policy-mce-hcp-autoimport
  namespace: open-cluster-management-global-set
  annotations:
    policy.open-cluster-management.io/standards: NIST SP 800-53
    policy.open-cluster-management.io/categories: CM Configuration Management
    policy.open-cluster-management.io/controls: CM-2 Baseline Configuration
    policy.open-cluster-management.io/description: Discovered clusters that are of
      type MultiClusterEngineHCP can be automatically imported into ACM as managed
clusters.
      This policy configure those discovered clusters so they are automatically imported.
      Fine tuning MultiClusterEngineHCP clusters to be automatically imported
      can be done by configure filters at the configMap or add annotation to the discoverd
cluster.
spec:
  disabled: false
  policy-templates:
```

```
  - objectDefinition:
      apiVersion: policy.open-cluster-management.io/v1
      kind: ConfigurationPolicy
      metadata:
        name: mce-hcp-autoimport-config
      spec:
        object-templates:
          - complianceType: musthave
            objectDefinition:
              apiVersion: v1
              kind: ConfigMap
              metadata:
                name: discovery-config
                namespace: open-cluster-management-global-set
              data:
                rosa-filter: ""
        remediationAction: enforce 1
        severity: low
  - objectDefinition:
      apiVersion: policy.open-cluster-management.io/v1
      kind: ConfigurationPolicy
      metadata:
        name: policy-mce-hcp-autoimport
      spec:
        remediationAction: enforce
        severity: low
        object-templates-raw: |
          {{- /* find the MultiClusterEngineHCP DiscoveredClusters */ -}}
          {{- range $dc := (lookup "discovery.open-cluster-management.io/v1"
"DiscoveredCluster" "" "").items }}
            {{- /* Check for the flag that indicates the import should be skipped */ -}}
            {{- $skip := "false" -}}
            {{- range $key, $value := $dc.metadata.annotations }}
              {{- if and (eq $key "discovery.open-cluster-management.io/previously-auto-
imported")
                         (eq $value "true") }}
                {{- $skip = "true" }}
              {{- end }}
            {{- end }}
            {{- /* if the type is MultiClusterEngineHCP and the status is Active */ -}}
            {{- if and (eq $dc.spec.status "Active")
                       (contains (fromConfigMap "open-cluster-management-global-set" "discovery-
config" "mce-hcp-filter") $dc.spec.displayName)
                       (eq $dc.spec.type "MultiClusterEngineHCP")
                       (eq $skip "false") }}
          - complianceType: musthave
            objectDefinition:
              apiVersion: discovery.open-cluster-management.io/v1
              kind: DiscoveredCluster
              metadata:
                name: {{ $dc.metadata.name }}
                namespace: {{ $dc.metadata.namespace }}
              spec:
                importAsManagedCluster: true 2
            {{- end }}
          {{- end }}
```

1. To enable automatic import, change the **spec.remediationAction** to **enforce**.
2. To enable automatic import, change **spec.importAsManagedCluster** to **true**.

2. Run **oc apply -f <filename>.yaml -n <namespace>** to apply the file.

1.2.3. Creating the placement definition

You need to create a placement definition that specifies the managed cluster for the policy deployment. Complete the following procedure:

1. Create the **Placement** definition that selects only the **local-cluster**, which is a hub cluster that is managed. Use the following YAML sample:

```
apiVersion: cluster.open-cluster-management.io/v1beta1
kind: Placement
metadata:
  name: policy-mce-hcp-autoimport-placement
  namespace: open-cluster-management-global-set
spec:
  tolerations:
    - key: cluster.open-cluster-management.io/unreachable
      operator: Exists
    - key: cluster.open-cluster-management.io/unavailable
      operator: Exists
  clusterSets:
    - global
  predicates:
    - requiredClusterSelector:
        labelSelector:
          matchExpressions:
            - key: local-cluster
              operator: In
              values:
                - "true"
```

2. Run **oc apply -f placement.yaml -n <namespace>**, where **namespace** matches the namespace that you used for the policy that you previously created.

1.2.4. Binding the import policy to a placement definition

After you create the policy and the placement, you need to connect the two resources. Complete the following steps:

1. Connect the resources by using a **PlacementBinding** resource. See the following example where **placementRef** points to the **Placement** that you created, and **subjects** points to the **Policy** that you created:

```
apiVersion: policy.open-cluster-management.io/v1
kind: PlacementBinding
metadata:
  name: policy-mce-hcp-autoimport-placement-binding
  namespace: open-cluster-management-global-set
placementRef:
```

```
  name: policy-mce-hcp-autoimport-placement
  apiGroup: cluster.open-cluster-management.io
  kind: Placement
subjects:
  - name: policy-mce-hcp-autoimport
    apiGroup: policy.open-cluster-management.io
    kind: Policy
```

2. To verify, run the following command:

```
oc get policy policy-mce-hcp-autoimport -n <namespace>
```

Important: You can *detach* a hosted cluster from Red Hat Advanced Cluster Management by using the **Detach** option in the Red Hat Advanced Cluster Management console, or by removing the corresponding **ManagedCluster** custom resource from the command line.

For best results, detach the managed hosted cluster before *destroying* the hosted cluster.

When a discovered cluster is detached, the following annotation is added to the **DiscoveredCluster** resource to prevent the policy to import the discovered cluster again.

```
annotations:
  discovery.open-cluster-management.io/previously-auto-imported: "true"
```

If you want the detached discovered cluster to be reimported, remove this annotation.

1.3. AUTOMATING IMPORT FOR DISCOVERED OPENSHIFT SERVICE ON AWS CLUSTERS

Automate the import of OpenShift Service on AWS clusters by using Red Hat Advanced Cluster Management policy enforcement for faster cluster management, without manually importing individual clusters.

Required access: Cluster administrator

- Prerequisites
- Creating the automatic import policy
- Creating the placement definition
- Binding the import policy to a placement definition

1.3.1. Prerequisites

- You need Red Hat Advanced Cluster Management installed. See the Red Hat Advanced Cluster Management Installing and upgrading documentation.
- You need to learn about *Policies*. See the introduction to Governance in the Red Hat Advanced Cluster Management documentation.

1.3.2. Creating the automatic import policy

The following policy and procedure is an example of how to import all your discovered OpenShift Service on AWS clusters automatically.

Log in to your hub cluster from the CLI to complete the following procedure:

1. Create a YAML file with the following example and apply the changes that are referenced:

```
apiVersion: policy.open-cluster-management.io/v1
kind: Policy
metadata:
  name: policy-rosa-autoimport
  annotations:
    policy.open-cluster-management.io/standards: NIST SP 800-53
    policy.open-cluster-management.io/categories: CM Configuration Management
    policy.open-cluster-management.io/controls: CM-2 Baseline Configuration
    policy.open-cluster-management.io/description: OpenShift Service on AWS discovered
clusters can be automatically imported into
Red Hat Advanced Cluster Management as managed clusters with this policy. You can
select and configure those managed clusters so you can import. Configure filters or add an
annotation if you do not want all of your OpenShift Service on AWS clusters to be
automatically imported.
spec:
  remediationAction: inform 1
  disabled: false
  policy-templates:
    - objectDefinition:
        apiVersion: policy.open-cluster-management.io/v1
        kind: ConfigurationPolicy
        metadata:
          name: rosa-autoimport-config
        spec:
          object-templates:
            - complianceType: musthave
              objectDefinition:
                apiVersion: v1
                kind: ConfigMap
                metadata:
                  name: discovery-config
                  namespace: open-cluster-management-global-set
                data:
                  rosa-filter: "" 2
          remediationAction: enforce
          severity: low
    - objectDefinition:
        apiVersion: policy.open-cluster-management.io/v1
        kind: ConfigurationPolicy
        metadata:
          name: policy-rosa-autoimport
        spec:
          remediationAction: enforce
          severity: low
          object-templates-raw: |
            {{- /* find the ROSA DiscoveredClusters */ -}}
            {{- range $dc := (lookup "discovery.open-cluster-management.io/v1"
"DiscoveredCluster" "" "").items }}
              {{- /* Check for the flag that indicates the import should be skipped */ -}}
```

```
              {{- $skip := "false" -}}
              {{- range $key, $value := $dc.metadata.annotations }}
                {{- if and (eq $key "discovery.open-cluster-management.io/previously-auto-
imported")
                           (eq $value "true") }}
                  {{- $skip = "true" }}
                {{- end }}
              {{- end }}
              {{- /* if the type is ROSA and the status is Active */ -}}
              {{- if and (eq $dc.spec.status "Active")
                         (contains (fromConfigMap "open-cluster-management-global-set" "discovery-
config" "rosa-filter") $dc.spec.displayName)
                         (eq $dc.spec.type "ROSA")
                         (eq $skip "false") }}
            - complianceType: musthave
              objectDefinition:
                apiVersion: discovery.open-cluster-management.io/v1
                kind: DiscoveredCluster
                metadata:
                  name: {{ $dc.metadata.name }}
                  namespace: {{ $dc.metadata.namespace }}
                spec:
                  importAsManagedCluster: true
              {{- end }}
            {{- end }}
    - objectDefinition:
        apiVersion: policy.open-cluster-management.io/v1
        kind: ConfigurationPolicy
        metadata:
          name: policy-rosa-managedcluster-status
        spec:
          remediationAction: enforce
          severity: low
          object-templates-raw: |
            {{- /* Use the same DiscoveredCluster list to check ManagedCluster status */ -}}
            {{- range $dc := (lookup "discovery.open-cluster-management.io/v1"
"DiscoveredCluster" "" "").items }}
              {{- /* Check for the flag that indicates the import should be skipped */ -}}
              {{- $skip := "false" -}}
              {{- range $key, $value := $dc.metadata.annotations }}
                {{- if and (eq $key "discovery.open-cluster-management.io/previously-auto-
imported")
                           (eq $value "true") }}
                  {{- $skip = "true" }}
                {{- end }}
              {{- end }}
              {{- /* if the type is ROSA and the status is Active */ -}}
              {{- if and (eq $dc.spec.status "Active")
                         (contains (fromConfigMap "open-cluster-management-global-set" "discovery-
config" "rosa-filter") $dc.spec.displayName)
                         (eq $dc.spec.type "ROSA")
                         (eq $skip "false") }}
            - complianceType: musthave
              objectDefinition:
                apiVersion: cluster.open-cluster-management.io/v1
                kind: ManagedCluster
```

```
          metadata:
            name: {{ $dc.spec.displayName }}
            namespace: {{ $dc.spec.displayName }}
          status:
            conditions:
              - type: ManagedClusterConditionAvailable
                status: "True"
        {{- end }}
      {{- end }}
```

1. To enable automatic import, change the **spec.remediationAction** to **enforce**.

2. Optional: Specify a value here to select a subset of the matching OpenShift Service on AWS clusters, which are based on *discovered* cluster names. The **rosa-filter** has no value by default, so the filter does not restrict cluster names without a subset value.

2. Run **oc apply -f <filename>.yaml -n <namespace>** to apply the file.

1.3.3. Creating the placement definition

You need to create a placement definition that specifies the managed cluster for the policy deployment.

1. Create the placement definition that selects only the **local-cluster**, which is a hub cluster that is managed. Use the following YAML sample:

```
apiVersion: cluster.open-cluster-management.io/v1beta1
kind: Placement
metadata:
  name: placement-openshift-plus-hub
spec:
  predicates:
  - requiredClusterSelector:
      labelSelector:
        matchExpressions:
        - key: name
          operator: In
          values:
          - local-cluster
```

2. Run **oc apply -f placement.yaml -n <namespace>**, where **namespace** matches the namespace that you used for the policy that you previously created.

1.3.4. Binding the import policy to a placement definition

After you create the policy and the placement, you need to connect the two resources.

1. Connect the resources by using a **PlacementBinding**. See the following example where **placementRef** points to the **Placement** that you created, and **subjects** points to the **Policy** that you created:

```
apiVersion: policy.open-cluster-management.io/v1
kind: PlacementBinding
metadata:
  name: binding-policy-rosa-autoimport
```

```
placementRef:
  apiGroup: cluster.open-cluster-management.io
  kind: Placement
  name: placement-policy-rosa-autoimport
subjects:
- apiGroup: policy.open-cluster-management.io
  kind: Policy
  name: policy-rosa-autoimport
```

2. To verify, run the following command:

```
oc get policy policy-rosa-autoimport -n <namespace>
```

1.4. OBSERVABILITY INTEGRATION

With the Red Hat Advanced Cluster Management Observability feature, you can view health and utilization of clusters across your fleet. You can install Red Hat Advanced Cluster Management and enable Observability.

1.4.1. Observing hosted control planes

After you enable the **multicluster-observability** pod, you can use Red Hat Advanced Cluster Management Observability Grafana dashboards to view the following information about your hosted control planes:

- **ACM** > **Hosted Control Planes Overview** dashboard to see cluster capacity estimates for hosting hosted control planes, the related cluster resources, and the list and status of existing hosted control planes. For more information, see: Introduction to hosted control planes .
- **ACM** > **Resources** > **Hosted Control Plane** dashboard that you can access from the *Overview* page to see the resource utilization of the selected hosted control plane. For more information, see Installing the hosted control planes command-line interface .

To enable, see Observability service.

CHAPTER 2. SITECONFIG OPERATOR

The SiteConfig operator offers a template-driven cluster provisioning solution, which allows you to provision clusters with various installation methods.

The SiteConfig operator introduces the unified **ClusterInstance** API, which comes from the **SiteConfig** API of the **SiteConfig** generator kustomize plugin.

The **ClusterInstance** API decouples parameters that define a cluster from the manner in which the cluster is deployed.

This separation removes certain limitations that are presented by the **SiteConfig** kustomize plugin in the current GitOps Zero Touch Provisioning (ZTP) flow , such as agent cluster installations and scalability constraints that are related to Argo CD.

Using the unified **ClusterInstance** API, the SiteConfig operator offers the following improvements:

Isolation

Separates the cluster definition from the installation method. The **ClusterInstance** custom resource captures the cluster definition, while installation templates capture the cluster architecture and installation methods.

Unification

The SiteConfig operator unifies both Git and non-Git workflows. You can apply the **ClusterInstance** custom resource directly on the hub cluster, or synchronize resources through a GitOps solution, such as ArgoCD.

Consistency

Maintains a consistent API across installation methods, whether you are using the Assisted Installer, the Image Based Install Operator, or any other custom template-based approach.

Scalability

Achieves greater scalability for each cluster than the **SiteConfig** kustomize plugin.

Flexibility

Provides you with more power to deploy and install clusters by using custom templates.

Troubleshooting

Offers insightful information regarding cluster deployment status and rendered manifests, significantly enhancing the troubleshooting experience.

For more information about the Image Based Install Operator, see Image Based Install Operator.

For more information about the Assisted Installer, see Installing an on-premise cluster using the Assisted Installer

2.1. THE SITECONFIG OPERATOR FLOW

The SiteConfig operator dynamically generates installation manifests based on user-defined templates that are instantiated from the data in the **ClusterInstance** custom resource.

You can source the **ClusterInstance** custom resource from your Git repository through ArgoCD, or you can create it directly on the hub cluster manually or through external tools and workflows.

The following is a high-level overview of the process:

1. You create one or more sets of installation templates on the hub cluster.

2. You create a **ClusterInstance** custom resource that references those installation templates and supporting manifests.

3. After the resources are created, the SiteConfig operator reconciles the **ClusterInstance** custom resource by populating the templated fields that are referenced in the custom resource.

4. The SiteConfig operator validates and renders the installation manifests, then the Operator performs a dry run.

5. If the dry run is successful, the manifests are created, then the underlying Operators consume and process the manifests.

6. The installation begins.

7. The SiteConfig operator continuously monitors for changes in the associated **ClusterDeployment** resource and updates the **ClusterInstance** custom resource's **status** field accordingly.

To learn more about how to use SiteConfig operator, see the following documentation:

- Installation templates overview
- Enabling the SiteConfig operator
- Image Based Install Operator
- Installing single-node OpenShift clusters with the SiteConfig operator
- Deprovisioning single-node OpenShift clusters with the SiteConfig operator
- Creating custom templates with the SiteConfig operator

For advanced topics, see the following documentation:

- Scaling in worker nodes with the SiteConfig operator
- Scaling out worker nodes with the SiteConfig operator

2.2. INSTALLATION TEMPLATES OVERVIEW

Installation templates are data-driven templates that are used to generate the set of installation artifacts. These templates follow the Golang **text/template** format, and are instantiated by using data from the **ClusterInstance** custom resource. This enables dynamic creation of installation manifests for each target cluster that has similar configurations, but with different values.

You can also create multiple sets based on the different installation methods or cluster topologies. The SiteConfig operator supports the following types of installation templates:

Cluster-level

Templates that must reference only cluster-specific fields.

Node-level

Templates that can reference both cluster-specific and node-specific fields.

For more information about installation templates, see the following documentation:

- Template functions

- Default set of templates
- Special template variables
- Customization of the manifests order

2.2.1. Template functions

You can customize the templated fields. The SiteConfig operator supports all sprig library functions.

Additionally, the **ClusterInstance** API provides the following function that you can use while creating your custom manifests:

toYaml

The **toYaml** function encodes an item into a YAML string. If the item cannot be converted to YAML, the function returns an empty string.

See the following example of the **.toYaml** specification in the **ClusterInstance.Spec.Proxy** field:

```
{{ if .Spec.Proxy }}
  proxy:
{{ .Spec.Proxy | toYaml | indent 4 }}
{{ end }}
```

2.2.2. Default set of templates

The SiteConfig operator provides the following default, validated, and immutable set of templates in the same namespace in which the operator is installed:

Installation method	Template type	File name	Template content
Assisted Installer	Cluster-level templates	**ai-cluster-templates-v1.yaml**	**AgentClusterInstall** **ClusterDeployment** **InfraEnv** **KlusterletAddonConfig** **ManagedCluster**
	Node-level templates	**ai-node-templates-v1.yaml**	**BareMetalHost** **NMStateConfig**
Image-based Install Operator	Cluster-level templates	**ibi-cluster-templates-v1.yaml**	**ClusterDeployment** **KlusterletAddonConfig** **ManagedCluster**
	Node-level templates	**ibi-node-templates-v1.yaml**	**BareMetalHost** **ImageClusterInstall** **NetworkSecret**

2.2.3. Special template variables

The SiteConfig operator provides a set of special template variables that you can use in your templates. See the following list:

CurrentNode
: The SiteConfig operator explicitly controls the iteration of the node objects and exposes this variable to access all the content for the current node being handled in templating.

InstallConfigOverrides
: Contains the merged **networkType**, **cpuPartitioningMode** and **installConfigOverrides** content.

ControlPlaneAgents
: Consists of the number of control plane agents and it is automatically derived from the **ClusterInstance** node objects.

WorkerAgents
: Consists of the number of worker agents and it is automatically derived from the **ClusterInstance** node objects.

Capitalize the field name in the text template to create a custom templated field.

For example, the **ClusterInstance spec** field is referenced with the **.Spec** prefix. However, you must reference special variable fields with the **.SpecialVars** prefix.

Important: Instead of using the **.Spec.Nodes** prefix for the **spec.nodes** field, you must reference it with the **.SpecialVars.CurrentNode** special template variable.

For example, if you want to specify the **name** and **namespace** for your current node by using the **CurrentNode** special template variable, use the field names in the following form:

```
name: "{{ .SpecialVars.CurrentNode.HostName }}"
namespace: "{{ .Spec.ClusterName }}"
```

2.2.4. Customization of the manifests order

You can control the order in which manifests are created, updated, and deleted by using the **siteconfig.open-cluster-management.io/sync-wave** annotation. The annotation takes an integer as a value, and that integer constitutes as a *wave*.

You can add one or several manifests to a single wave. If you do not specify a value, the annotation takes the default value of **0**.

The SiteConfig operator reconciles the manifests in ascending order when creating or updating resources and it deletes resources in descending order.

In the following example, if the SiteConfig operator creates or updates the manifests, the **AgentClusterInstall** and **ClusterDeployment** custom resources are reconciled in the first wave, while **KlusterletAddonConfig** and **ManagedCluster** custom resources are reconciled in the third wave.

```
apiVersion: v1
data:
  AgentClusterInstall: |-
    ...
      siteconfig.open-cluster-management.io/sync-wave: "1"
    ...
  ClusterDeployment: |-
    ...
```

```
      siteconfig.open-cluster-management.io/sync-wave: "1"
   ...
  InfraEnv: |-
   ...
      siteconfig.open-cluster-management.io/sync-wave: "2"
   ...
  KlusterletAddonConfig: |-
   ...
      siteconfig.open-cluster-management.io/sync-wave: "3"
   ...
  ManagedCluster: |-
   ...
      siteconfig.open-cluster-management.io/sync-wave: "3"
   ...
kind: ConfigMap
metadata:
  name: assisted-installer-templates
  namespace: example-namespace
```

If the SiteConfig operator deletes the resources, **KlusterletAddonConfig** and **ManagedCluster** custom resources are the first to be deleted, while the **AgentClusterInstall** and **ClusterDeployment** custom resources are the last.

2.3. ENABLING THE SITECONFIG OPERATOR

Enable the SiteConfig operator to use the default installation templates and install single-node OpenShift clusters at scale.

Required access: Cluster administrator

2.3.1. Prerequisites

- You need a Red Hat Advanced Cluster Management version 2.12 hub cluster.

2.3.2. Enabling the SiteConfig operator from the MultiClusterHub resource

Patch the **MultiClusterHub** resource, then verify that SiteConfig operator is enabled. Complete the following procedure:

1. Set the **enabled** field to **true** in the **siteconfig** entry of **spec.overrides.components** in the **Multiclusterhub** resource by running the following command:

```
oc patch multiclusterhubs.operator.open-cluster-management.io multiclusterhub -n rhacm --
type json --patch '[{"op": "add", "path":"/spec/overrides/components/-", "value":
{"name":"siteconfig","enabled": true}}]'
```

2. Verify that the SiteConfig operator is enabled by running the following command on the hub cluster:

```
oc -n rhacm get po | grep siteconfig
```

See the following example output:

```
siteconfig-controller-manager-6fdd86cc64-sdg87          2/2     Running   0          43s
```

–

3. **Optional:** Verify that you have the default installation templates by running the following command on the hub cluster:

```
oc -n rhacm get cm
```

See the following list of templates in the output example:

```
NAME                         DATA   AGE
ai-cluster-templates-v1        5     97s
ai-node-templates-v1           2     97s
...
ibi-cluster-templates-v1       3     97s
ibi-node-templates-v1          3     97s
...
```

2.4. IMAGE BASED INSTALL OPERATOR

Install the Image Based Install Operator so that you can complete and manage image-based cluster installations by using the same APIs as existing installation methods.

For more information about, and to learn how to enable the Image Based Install Operator, see Image-based installations for single-node OpenShift.

2.5. INSTALLING SINGLE-NODE OPENSHIFT CLUSTERS WITH THE SITECONFIG OPERATOR

Install your clusters with the SiteConfig operator by using the default installation templates. Use the installation templates for the Image-Based Install Operator to complete the procedure.

Required access: Cluster administrator

2.5.1. Prerequisites

- If you are using GitOps ZTP, configure your GitOps ZTP environment. To configure your environment, see Preparing the hub cluster for GitOps ZTP .
- You have the default installation templates. To get familiar with the default templates, see Default set of templates
- Install and configure the underlying operator of your choice.
  - To learn about and install the Image Based Install Operator for single-node OpenShift, see Image Based Install Operator.
  - To install the Assisted Installer, see Installing an on-premise cluster with the Assisted Installer.

Complete the following steps to install a cluster with the SiteConfig operator:

1. Creating the target namespace
2. Creating the pull secret

3. Creating the BMC secret
4. Optional: Creating the extra manifests
5. Rendering the installation manifests

2.5.2. Creating the target namespace

You need a target namespace when you create the pull secret, the BMC secret, extra manifest **ConfigMap** objects, and the **ClusterInstance** custom resource.

Complete the following steps to create the target namespace:

1. Create a YAML file for the target namespace. See the following example file that is named **clusterinstance-namespace.yaml**:

```
apiVersion: v1
kind: Namespace
metadata:
  name: example-sno
```

2. Apply your file to create the resource. Run the following command on the hub cluster:

```
oc apply -f clusterinstance-namespace.yaml
```

2.5.3. Creating the pull secret

You need a pull secret to enable your clusters to pull images from container registries. Complete the following steps to create a pull secret:

1. Create a YAML file to pull images. See the following example of a file that is named **pull-secret.yaml**:

```
apiVersion: v1
kind: Secret
metadata:
  name: pull-secret
  namespace: example-sno 1
data:
  .dockerconfigjson: <encoded_docker_configuration> 2
type: kubernetes.io/dockerconfigjson
```

1 The **namespace** value must match the target namespace.

2 Specify the base64-encoded configuration file as the value.

2. Apply the file to create the resource. Run the following command on the hub cluster:

```
oc apply -f pull-secret.yaml
```

2.5.4. Creating the BMC secret

You need a secret to connect to your baseboard management controller (BMC). Complete the following steps to create a secret:

1. Create a YAML file for the BMC secret. See the following sample file that is named **example-bmc-secret.yaml**:

```
apiVersion: v1
data:
  password: <password>
  username: <username>
kind: Secret
metadata:
  name: example-bmh-secret
  namespace: "example-sno" 1
type: Opaque
```

1 The **namespace** value must match the target namespace.

2. Apply the file to create the resource. Run the following command on the hub cluster:

```
oc apply -f example-bmc-secret.yaml
```

2.5.5. Optional: Creating the extra manifests

You can create extra manifests that you need to reference in the **ClusterInstance** custom resource. Complete the following steps to create an extra manifest:

1. Create a YAML file for an extra manifest **ConfigMap** object, for example named **enable-crun.yaml**:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: enable-crun
  namespace: example-sno 1
data:
  enable-crun-master.yaml: |
    apiVersion: machineconfiguration.openshift.io/v1
    kind: ContainerRuntimeConfig
    metadata:
      name: enable-crun-master
    spec:
      machineConfigPoolSelector:
        matchLabels:
          pools.operator.machineconfiguration.openshift.io/master: ""
      containerRuntimeConfig:
        defaultRuntime: crun
  enable-crun-worker.yaml: |
    apiVersion: machineconfiguration.openshift.io/v1
    kind: ContainerRuntimeConfig
    metadata:
      name: enable-crun-worker
    spec:
      machineConfigPoolSelector:
```

```
    matchLabels:
      pools.operator.machineconfiguration.openshift.io/worker: ""
  containerRuntimeConfig:
    defaultRuntime: crun
```

1. The **namespace** value must match the target namespace.

2. Create the resource by running the following command on the hub cluster:

```
oc apply -f enable-crun.yaml
```

2.5.6. Rendering the installation manifests

Reference the templates and supporting manifests in the **ClusterInstance** custom resource. Complete the following steps to render the installation manifests by using the default cluster and node templates:

1. In the **example-sno** namespace, create the **ClusterInstance** custom resource that is named **clusterinstance-ibi.yaml** in the following example:

```
apiVersion: siteconfig.open-cluster-management.io/v1alpha1
kind: ClusterInstance
metadata:
  name: "example-clusterinstance"
  namespace: "example-sno" 1
spec:
  holdInstallation: false
  extraManifestsRefs: 2
    - name: extra-machine-configs
    - name: enable-crun
  pullSecretRef:
    name: "pull-secret" 3
  [...]
  templateRefs: 4
    - name: ibi-cluster-templates-v1
      namespace: rhacm
  [...]
  nodes:
      [...]
      bmcCredentialsName: 5
        name: "example-bmh-secret"
      [...]
      templateRefs: 6
        - name: ibi-node-templates-v1
          namespace: rhacm
      [...]
```

1. The **namespace** in the **ClusterInstance** custom resource must match the target namespace that you defined.

2. Reference the **name** of one or more extra manifests **ConfigMap** objects.

3. Reference the **name** of your pull secret.

4. Reference the **name** of the cluster-level templates under the **spec.templateRefs** field. The **namespace** must match the namespace where the Operator is installed.

5. Reference the **name** of the BMC secret.

6. Reference the **name** of the node-level templates under the **spec.nodes.templateRefs** field. The **namespace** must match the namespace where the Operator is installed.

2. Apply the file and create the resource by running the following command:

```
oc apply -f clusterinstance-ibi.yaml
```

After you create the custom resource, the SiteConfig operator starts reconciling the **ClusterInstance** custom resource, then validates and renders the installation manifests.

The SiteConfig operator continues to monitor for changes in the **ClusterDeployment** custom resources to update the cluster installation progress of the corresponding **ClusterInstance** custom resource.

3. Monitor the process by running the following command:

```
oc get clusterinstance <cluster_name> -n <target_namespace> -o yaml
```

See the following example output from the **status.conditions** section for successful manifest generation:

```
message: Applied site config manifests
reason: Completed
status: "True"
type: RenderedTemplatesApplied
```

4. Check the manifests that SiteConfig operator rendered by running the following command:

```
oc get clusterinstance <cluster_name> -n <target_namespace> -o
jsonpath='{.status.manifestsRendered}'
```

For more information about status conditions, see ClusterInstance API.

2.6. DEPROVISIONING SINGLE-NODE OPENSHIFT CLUSTERS WITH THE SITECONFIG OPERATOR

Deprovision your clusters with the SiteConfig operator to delete all resources and accesses associated with that cluster.

Required access: Cluster administrator

2.6.1. Prerequisites

- Deploy your clusters with the SiteConfig operator by using the default installation templates.

2.6.2. Deprovisioning single-node OpenShift clusters

Complete the following steps to delete your clusters:

1. Delete the **ClusterInstance** custom resource by running the following command:

```
oc delete clusterinstance <cluster_name> -n <target_namespace>
```

2. Verify that the deletion was successful by running the following command:

```
oc get clusterinstance <cluster_name> -n <target_namespace>
```

See the following example output where the **(NotFound)** error indicates that your cluster is deprovisioned.

```
Error from server (NotFound): clusterinstances.siteconfig.open-cluster-management.io "
<cluster_name>" not found
```

2.7. CREATING CUSTOM TEMPLATES WITH THE SITECONFIG OPERATOR

Create user-defined templates that are not provided in the default set of templates.

Required access: Cluster administrator

Complete the following steps to a create custom template:

1. Create a YAML file named **my-custom-secret.yaml** that contains the cluster-level template in a **ConfigMap** object:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: my-custom-secret
  namespace: rhacm
data:
  MySecret: |-
    apiVersion: v1
    kind: Secret
    metadata:
      name: "{{ .Spec.ClusterName }}-my-custom-secret-key"
      namespace: "clusters"
      annotations:
        siteconfig.open-cluster-management.io/sync-wave: "1" 1
    type: Opaque
    data:
      key: <key>
```

1 The **siteconfig.open-cluster-management.io/sync-wave** annotation controls in which order manifests are created, updated, or deleted.

2. Apply the custom template on the hub cluster by running the following command:

```
oc apply -f my-custom-secret.yaml
```

3. Reference your template in the **ClusterInstance** custom resource named **clusterinstance-my-custom-secret.yaml**:

```
spec:
  ...
  templateRefs:
    - name: ai-cluster-templates-v1.yaml
      namespace: rhacm
    - name: my-custom-secret.yaml
      namespace: rhacm
    ...
```

4. Apply the **ClusterInstance** custom resource by running the following command:

```
oc apply -f clusterinstance-my-custom-secret.yaml
```

2.8. SCALING IN A SINGLE-NODE OPENSHIFT CLUSTER WITH THE SITECONFIG OPERATOR

Scale in your managed cluster that was installed by the SiteConfig operator. You can scale in your cluster by removing a worker node.

Required access: Cluster administrator

2.8.1. Prerequisites

- If you are using GitOps ZTP, you have configured your GitOps ZTP environment. To configure your environment, see Preparing the hub cluster for GitOps ZTP .
- You have the default templates. To get familiar with the default templates, see Default set of templates
- You have installed your cluster with the SiteConfig operator. To install a cluster with the SiteConfig operator, see Installing single-node OpenShift clusters with the SiteConfig operator

2.8.2. Adding an annotation to your worker node

Add an annotation to your worker node for removal.

Complete the following steps to annotate worker node from the managed cluster:

1. Add an annotation in the **extraAnnotations** field of the worker node entry in the **ClusterInstance** custom resource that is used to provision your cluster:

```
spec:
  ...
  nodes:
  - hostName: "worker-node2.example.com"
    role: "worker"
    ironicInspect: ""
    extraAnnotations:
      BareMetalHost:
        bmac.agent-install.openshift.io/remove-agent-and-node-on-delete: "true"
...
```

–

2. Apply the changes. See the following options:

   a. If you are using Red Hat Advanced Cluster Management without Red Hat OpenShift GitOps, run the following command on the hub cluster:

   ```
   oc apply -f <clusterinstance>.yaml
   ```

   a. If you are using GitOps ZTP, push to your Git repository and wait for Argo CD to synchronize the changes.

3. Verify that the annotation is applied to the **BaremetalHost** worker resource by running the following command on the hub cluster:

   ```
   oc get bmh -n <clusterinstance_namespace> worker-node2.example.com -
   ojsonpath='{.metadata.annotations}' | jq
   ```

   See the following example output for successful application of the annotation:

```
{
  "baremetalhost.metal3.io/detached": "assisted-service-controller",
  "bmac.agent-install.openshift.io/hostname": "worker-node2.example.com",
  "bmac.agent-install.openshift.io/remove-agent-and-node-on-delete": "true"
  "bmac.agent-install.openshift.io/role": "master",
  "inspect.metal3.io": "disabled",
  "siteconfig.open-cluster-management.io/sync-wave": "1",
}
```

2.8.3. Deleting the BareMetalHost resource of the worker node

Delete the **BareMetalHost** resource of the worker node that you want to be removed.

Complete the following steps to remove a worker node from the managed cluster:

1. Update the node object that you want to delete in your existing **ClusterInstance** custom resource with the following configuration:

   ```
   ...
   spec:
     ...
     nodes:
      - hostName: "worker-node2.example.com"
        ...
        pruneManifests:
          - apiVersion: metal3.io/v1alpha1
            kind: BareMetalHost
   ...
   ```

2. Apply the changes. See the following options.

   a. If you are using Red Hat Advanced Cluster Management without Red Hat OpenShift GitOps, run the following command on the hub cluster:

   ```
   oc apply -f <clusterinstance>.yaml
   ```

a. If you are using GitOps ZTP, push to your Git repository and wait for Argo CD to synchronize the changes.

3. Verify that the **BareMetalHost** resources are removed by running the following command on the hub cluster:

```
oc get bmh -n <clusterinstance_namespace> --watch --kubeconfig
<hub_cluster_kubeconfig_filename>
```

See the following example output:

```
NAME                    STATE                   CONSUMER        ONLINE   ERROR   AGE
master-node1.example.com   provisioned                 true          81m
worker-node2.example.com   deprovisioning              true          44m
worker-node2.example.com   powering off before delete   true            20h
worker-node2.example.com   deleting                  true          50m
```

4. Verify that the **Agent** resources are removed by running the following command on the hub cluster:

```
oc get agents -n <clusterinstance_namespace> --kubeconfig
<hub_cluster_kubeconfig_filename>
```

See the following example output:

```
NAME                    CLUSTER                APPROVED   ROLE     STAGE
master-node1.example.com   <managed_cluster_name>   true       master   Done
master-node2.example.com   <managed_cluster_name>   true       master   Done
master-node3.example.com   <managed_cluster_name>   true       master   Done
worker-node1.example.com   <managed_cluster_name>   true       worker   Done
```

5. Verify that the **Node** resources are removed by running the following command on the managed cluster:

```
oc get nodes --kubeconfig <managed_cluster_kubeconfig_filename>
```

See the following example output:

```
NAME                    STATUS                     ROLES                 AGE   VERSION
worker-node2.example.com   NotReady,SchedulingDisabled   worker                19h   v1.30.5
worker-node1.example.com   Ready                      worker                19h   v1.30.5
master-node1.example.com   Ready                      control-plane,master   19h   v1.30.5
master-node2.example.com   Ready                      control-plane,master   19h   v1.30.5
master-node3.example.com   Ready                      control-plane,master   19h   v1.30.5
```

2.9. SCALING OUT A SINGLE-NODE OPENSHIFT CLUSTER WITH THE SITECONFIG OPERATOR

Scale out your managed cluster that was installed by the SiteConfig operator. You can scale out your cluster by adding a worker node.

Required access: Cluster administrator

2.9.1. Prerequisites

- If using GitOps ZTP, you have configured your GitOps ZTP environment. To configure your environment, see Preparing the hub cluster for GitOps ZTP .
- You have the default installation templates. To get familiar with the default templates, see Default set of templates.
- You have installed your cluster with the SiteConfig operator. To install a cluster with the SiteConfig operator, see Installing single-node OpenShift clusters with the SiteConfig operator .

2.9.2. Adding a worker node

Add a worker node by updating your **ClusterInstance** custom resource that is used to provision your cluster.

Complete the following steps to add a worker node to the managed cluster:

1. Define a new node object in the existing **ClusterInstance** custom resource:

```
spec:
  ...
  nodes:
    - hostName: "<host_name>"
      role: "worker"
      templateRefs:
        - name: ai-node-templates-v1
          namespace: rhacm
      bmcAddress: "<bmc_address>"
      bmcCredentialsName:
        name: "<bmc_credentials_name>"
      bootMACAddress: "<boot_mac_address>"
...
```

2. Apply the changes. See the following options:

   a. If you are using Red Hat Advanced Cluster Management without Red Hat OpenShift GitOps, run the following command on the hub cluster:

```
oc apply -f <clusterinstance>.yaml
```

   a. If you are using GitOps ZTP, push to your Git repository and wait for Argo CD to synchronize the changes.

3. Verify that a new **BareMetalHost** resource is added by running the following command on the hub cluster:

```
oc get bmh -n <clusterinstance_namespace> --watch --kubeconfig
<hub_cluster_kubeconfig_filename>
```

See the following example output:

```
NAME                     STATE          CONSUMER   ONLINE   ERROR   AGE
master-node1.example.com  provisioned              true             81m
worker-node2.example.com  provisioning             true             44m
```

4. Verify that a new **Agent** resource is added by running the following command on the hub cluster:

```
oc get agents -n <clusterinstance_namespace> --kubeconfig
<hub_cluster_kubeconfig_filename>
```

See the following example output:

```
NAME                     CLUSTER                APPROVED   ROLE     STAGE
master-node1.example.com   <managed_cluster_name>   true       master   Done
master-node2.example.com   <managed_cluster_name>   true       master   Done
master-node3.example.com   <managed_cluster_name>   true       master   Done
worker-node1.example.com   <managed_cluster_name>   false      worker
worker-node2.example.com   <managed_cluster_name>   true       worker   Starting
installation
worker-node2.example.com   <managed_cluster_name>   true       worker   Installing
worker-node2.example.com   <managed_cluster_name>   true       worker   Writing image to
disk
worker-node2.example.com   <managed_cluster_name>   true       worker   Waiting for
control plane
worker-node2.example.com   <managed_cluster_name>   true       worker   Rebooting
worker-node2.example.com   <managed_cluster_name>   true       worker   Joined
worker-node2.example.com   <managed_cluster_name>   true       worker   Done
```

5. Verify that a new **Node** resource is added by running the following command on the managed cluster:

```
oc get nodes --kubeconfig <managed_cluster_kubeconfig_filename>
```

See the following example output:

```
NAME                     STATUS    ROLES                AGE   VERSION
worker-node2.example.com   Ready     worker               1h    v1.30.5
worker-node1.example.com   Ready     worker               19h   v1.30.5
master-node1.example.com   Ready     control-plane,master   19h   v1.30.5
master-node2.example.com   Ready     control-plane,master   19h   v1.30.5
master-node3.example.com   Ready     control-plane,master   19h   v1.30.5
```