Red Hat Advanced Cluster Management for Kubernetes 2.12

Governance

The Governance policy framework helps harden cluster security by using policies.

Last Updated: 2026-02-10

Red Hat Advanced Cluster Management for Kubernetes 2.12 Governance

The Governance policy framework helps harden cluster security by using policies.

Legal Notice

Abstract

Read more to learn about the governance policy framework, which helps harden cluster security by using policies.

Table of Contents

CHAPTER 1. GOVERNANCE

Enterprises must meet internal standards for software engineering, secure engineering, resiliency, security, and regulatory compliance for workloads hosted on private, multi and hybrid clouds. Red Hat Advanced Cluster Management for Kubernetes governance provides an extensible policy framework for enterprises to introduce their own security policies.

Continue reading the related topics of the Red Hat Advanced Cluster Management governance framework:

- Policy deployment
- Policy controllers
- Policy controller advanced configuration
- Configuring policy compliance history API (Technology Preview)
- Supported policies
- Policy dependencies
- Governance dashboard
- Securing the hub cluster
- Gatekeeper operator overview
- Integrating Policy Generator

1.1. POLICY CONTROLLERS

Policy controllers monitor and report whether your cluster is compliant with a policy. Use the Red Hat Advanced Cluster Management for Kubernetes policy framework by using the supported policy templates to apply policies managed by these controllers. The policy controllers manage Kubernetes custom resource definition instances.

Policy controllers check for policy violations, and can make the cluster status compliant if the controller supports the enforcement feature. View the following topics to learn more about the following Red Hat Advanced Cluster Management for Kubernetes policy controllers:

- Kubernetes configuration policy controller
- Certificate policy controller
- Policy set controller
- Operator policy controller

Important: Only the configuration policy controller policies support the **enforce** feature. You must manually remediate policies, where the policy controller does not support the **enforce** feature.

1.1.1. Kubernetes configuration policy controller

Use the configuration policy controller to configure any Kubernetes resource and apply security policies across your clusters. The configuration policy controller communicates with the local Kubernetes API server so that you can get a list of your configurations that are in your cluster.

During installation, the configuration policy controller is created on the managed cluster. The configuration policy is provided in the **policy-templates** field of the policy on the hub cluster, and is propagated to the selected managed clusters by the governance framework.

When the **remediationAction** for the configuration policy controller is set to **InformOnly**, the parent policy does not enforce the configuration policy, even if the **remediationAction** in the parent policy is set to **enforce**.

If you have existing Kubernetes manifests that you want to put in a policy, the Policy Generator is a useful tool to accomplish this.

1.1.1.1. Configuration policy YAML structure

You can find the description of a field on your managed cluster by running the **oc explain --api-version=policy.open-cluster-management.io/v1 ConfigurationPolicy.<field-path>** command. Replace **<field-path>** with the path to the field that you need.

```
apiVersion: policy.open-cluster-management.io/v1
kind: ConfigurationPolicy
metadata:
  name: policy-config
spec:
  namespaceSelector:
    include: ["default"]
    exclude: []
    matchExpressions: []
    matchLabels: {}
  remediationAction: inform (1)
  customMessage:
    compliant: {}
    noncompliant: {}
  severity: low
  evaluationInterval:
    compliant: ""
    noncompliant: ""
  object-templates-raw: ""
  object-templates: (2)
   - complianceType: musthave
     metadataComplianceType:
     recordDiff: ""
     recreateOption: ""
     objectSelector:
       matchLabels: {}
       matchExpressions: []
     objectDefinition:
       apiVersion: v1
       kind: Pod
       metadata:
         name: pod
       spec:
         containers:
```

```
        - image: pod-image
          name: pod-name
          ports:
           - containerPort: 80
  - complianceType: mustonlyhave
    objectDefinition:
      apiVersion: v1
      kind: ConfigMap
      metadata:
        name: myconfig
        namespace: default
        data:
          testData: hello
...
```

1. Configuration policies that specify an object without a name can only be set to **inform**. When the **remediationAction** for the configuration policy is set to **enforce**, the controller applies the specified configuration to the target managed cluster.

2. A Kubernetes object is defined in the **object-templates** array in the configuration policy, where fields of the configuration policy controller is compared with objects on the managed cluster. You can also use templated values within configuration policies. For more advanced use cases, specify a string in **object-templates-raw** to create the **object-templates** that you want. For more information, see *Template processing*.

1.1.1.2. Configuration policy YAML table

Table 1.1. Parameter table

Field	Optional or required	Description
apiVersion	Required	Set the value to **policy.open-cluster-management.io/v1**.
kind	Required	Set the value to **ConfigurationPolicy** to indicate the type of policy.
metadata.name	Required	The name of the policy.

Field	Optional or required	Description
spec.namespaceSelector	Required for namespaced objects that do not have a namespace specified	Determines namespaces in the managed cluster that the object is applied to. The **include** and **exclude** parameters accept file path expressions to include and exclude namespaces by name. The **matchExpressions** and **matchLabels** parameters specify namespaces to include by label. See the Kubernetes labels and selectors documentation. The resulting list is compiled by using the intersection of results from all parameters.
spec.remediationAction	Required	Specifies the action to take when the policy is non-compliant. Use the following parameter values: **inform**, **InformOnly**, or **enforce**.

Field	Optional or required	Description
spec.customMessage	Optional	Configure the compliance message sent by the configuration policy based on the current compliance. Each message configuration is a string that can contain Go templates. The **.DefaultMessage** and **.Policy** context variables are available for use in the templates. You can access the default message by using the **.DefaultMessage** parameter. The **.Policy** context variable contains the current policy object, including its status. For example, you can access the state of each related object by specifying the **.Policy.status.relatedObjects[*].object** field. If you set a value for the **evaluationInterval** field other than **watch**, only the kind, name, and namespace of the related objects are available. [source,yaml] ---- .Policy.status.relatedObjects[*].object ---- If you set an **evaluationInterval**, only identifiable information is available.
spec.customMessage.compliant	Optional	Configure custom messages for configuration policies that are compliant. Go templates and UTF-8 encoded characters, including emoji and foreign characters, are supported values.
spec.customMessage.noncompliant	Optional	Configure custom messages for configuration policies that are non-compliant. Go templates and UTF-8 encoded characters, including emoji and foreign characters, are supported values.
spec.severity	Required	Specifies the severity when the policy is non-compliant. Use the following parameter values: **low**, **medium**, **high**, or **critical**.

Field	Optional or required	Description
spec.evaluationInterval	Optional	Specifies the frequency for a policy to be evaluated when it is in a particular compliance state. Use the parameters **compliant** and **noncompliant**. The default value for the **compliant** and **noncompliant** parameters is **watch** to leverage Kubernetes API watches instead of polling the Kubernetes API server. When managed clusters have low resources, the evaluation interval can be set to long polling intervals to reduce CPU and memory usage on the Kubernetes API and policy controller. These are in the format of durations. For example, **1h25m3s** represents 1 hour, 25 minutes, and 3 seconds. These can also be set to **never** to avoid evaluating the policy after it is in a particular compliance state.
spec.evaluationInterval.compliant	Optional	Specifies the evaluation frequency for a compliant policy. To enable the previous polling behavior, set this parameter to **10s**.
spec.evaluationInterval.noncompliant	Optional	Specifies the evaluation frequency for a non-compliant policy. To enable the previous polling behavior, set this parameter to **10s**.
spec.object-templates	Optional	The array of Kubernetes objects (either fully defined or containing a subset of fields) for the controller to compare with objects on the managed cluster. **Note:** While **spec.object-templates** and **spec.object-templates-raw** are listed as optional, exactly one of the two parameter fields must be set.

Field	Optional or required	Description
spec.object-templates-raw	Optional	Used to set object templates with a raw YAML string. Specify conditions for the object templates, where advanced functions like if-else statements and the **range** function are supported values. For example, add the following value to avoid duplication in your **object-templates** definition: {{- if eq .metadata.name "policy-grc-your-meta-data-name" }} replicas: 2 {{- else }} replicas: 1 {{- end }} **Note:** While **spec.object-templates** and **spec.object-templates-raw** are listed as optional, exactly one of the two parameter fields must be set.

Field	Optional or required	Description
spec.object-templates[].complianceType	Required	Use this parameter to define the desired state of the Kubernetes object on your managed clusters. Use one of the following verbs as the parameter value: • **mustonlyhave**: Indicates that an object must exist with the exact fields and values as defined in the **objectDefinition**. • **musthave**: Indicates an object must exist with the same fields as specified in the **objectDefinition**. Any existing fields on the object that are not specified in the **object-template** are ignored. In general, array values are appended. The exception for the array to be patched is when the item contains a **name** key with a value that matches an existing item. Use a fully defined **objectDefinition** using the **mustonlyhave** compliance type, if you want to replace the array. • **mustnothave**: Indicates that an object with the same fields as specified in the **objectDefinition** cannot exist.
spec.object-templates[].metadataComplianceType	Optional	Overrides **spec.object-templates[].complianceType** when comparing the manifest's metadata section to objects on the cluster ("musthave", "mustonlyhave"). Default is unset to not override **complianceType** for metadata.

Field	Optional or required	Description
spec.object-templates[].recordDiff	Optional	Use this parameter to specify if and where to display the difference between the object on the cluster and the **objectDefinition** in the policy. The following options are supported: • Set to **InStatus** to store the difference in the **ConfigurationPolicy** status. • Set to **Log** to log the difference in the controller logs. • Set to **None** to not log the difference. By default, this parameter is set to **InStatus** if the controller does not detect sensitive data in the difference. Otherwise, the default is **None**. If sensitive data is detected, the **ConfigurationPolicy** status displays a message to set **recordDiff** to view the difference.
spec.object-templates[].recreateOption	Optional	Describes when to delete and recreate an object when an update is required. When you set the object to **IfRequired**, the policy recreates the object when updating an immutable field. When you set the parameter to **Always**, the policy recreates the object on any update. When you set the **remediationAction** to **inform**, the parameter value, **recreateOption**, has no effect on the object. The **IfRequired** value has no effect on clusters without dry-run update support. The default value is **None**.

Field	Optional or required	Description
spec.object-templates[].objectDefinition	Required	A Kubernetes object (either fully defined or containing a subset of fields) for the controller to compare with objects on the managed cluster.
spec.pruneObjectBehavior	Optional	Determines whether to clean up resources related to the policy when the policy is removed from a managed cluster.

1.1.1.3. Additional resources

See the following topics for more information:

- See Creating configuration policies.
- See the Hub cluster policy framework for more details on the hub cluster policy.
- See the policy samples that use NIST Special Publication 800-53 (Rev. 4) , and are supported by Red Hat Advanced Cluster Management from the **CM-Configuration-Management** folder.
- For information about dry-run support, see the Kubernetes documentation, Dry-run.
- Learn about how policies are applied on your hub cluster, see Supported policies for more details.
- Refer to Policy controllers for more details about controllers.
- Customize your policy controller configuration. See Policy controller advanced configuration .
- Learn about cleaning up resources and other topics in the Cleaning up resources that are created by policies documentation.
- Refer to Policy Generator.
- Learn about how to create and customize policies, see Governance dashboard.
- See Template processing.

1.1.2. Certificate policy controller

You can use the certificate policy controller to detect certificates that are close to expiring, time durations (hours) that are too long, or contain DNS names that fail to match specified patterns. You can add the certificate policy to the **policy-templates** field of the policy on the hub cluster, which propagates to the selected managed clusters by using the governance framework. See the Hub cluster policy framework documentation for more details on the hub cluster policy.

Configure and customize the certificate policy controller by updating the following parameters in your controller policy:

- **minimumDuration**

- **minimumCADuration**
- **maximumDuration**
- **maximumCADuration**
- **allowedSANPattern**
- **disallowedSANPattern**

Your policy might become non-compliant due to either of the following scenarios:

- When a certificate expires in less than the minimum duration of time or exceeds the maximum time.
- When DNS names fail to match the specified pattern.

The certificate policy controller is created on your managed cluster. The controller communicates with the local Kubernetes API server to get the list of secrets that contain certificates and determine all non-compliant certificates.

Certificate policy controller does not support the **enforce** feature.

Note: The certificate policy controller automatically looks for a certificate in a secret in only the **tls.crt** key. If a secret is stored under a different key, add a label named **certificate_key_name** with a value set to the key to let the certificate policy controller know to look in a different key. For example, if a secret contains a certificate stored in the key named **sensor-cert.pem**, add the following label to the secret: **certificate_key_name: sensor-cert.pem**.

1.1.2.1. Certificate policy controller YAML structure

View the following example of a certificate policy and review the element in the YAML table:

```
apiVersion: policy.open-cluster-management.io/v1
kind: CertificatePolicy
metadata:
  name: certificate-policy-example
spec:
  namespaceSelector:
    include: ["default"]
    exclude: []
    matchExpressions: []
    matchLabels: {}
  labelSelector:
    myLabelKey: myLabelValue
  remediationAction:
  severity:
  minimumDuration:
  minimumCADuration:
  maximumDuration:
  maximumCADuration:
  allowedSANPattern:
  disallowedSANPattern:
```

1.1.2.1.1. Certificate policy controller YAML table

Table 1.2. Parameter table

Field	Optional or required	Description
apiVersion	Required	Set the value to **policy.open-cluster-management.io/v1**.
kind	Required	Set the value to **CertificatePolicy** to indicate the type of policy.
metadata.name	Required	The name to identify the policy.
metadata.labels	Optional	In a certificate policy, the **category=system-and-information-integrity** label categorizes the policy and facilitates querying the certificate policies. If there is a different value for the **category** key in your certificate policy, the value is overridden by the certificate controller.
spec.namespaceSelector	Required	Determines namespaces in the managed cluster where secrets are monitored. The **include** and **exclude** parameters accept file path expressions to include and exclude namespaces by name. The **matchExpressions** and **matchLabels** parameters specify namespaces to be included by label. See the Kubernetes labels and selectors documentation. The resulting list is compiled by using the intersection of results from all parameters. **Note:** If the **namespaceSelector** for the certificate policy controller does not match any namespace, the policy is considered compliant.
spec.labelSelector	Optional	Specifies identifying attributes of objects. See the Kubernetes labels and selectors documentation.

Field	Optional or required	Description
spec.remediationAction	Required	Specifies the remediation of your policy. Set the parameter value to **inform**. Certificate policy controller only supports **inform** feature.
spec.severity	Optional	Informs the user of the severity when the policy is non-compliant. Use the following parameter values: **low**, **medium**, **high**, or **critical**.
spec.minimumDuration	Required	When a value is not specified, the default value is **100h**. This parameter specifies the smallest duration (in hours) before a certificate is considered non-compliant. The parameter value uses the time duration format from Golang. See Golang Parse Duration for more information.
spec.minimumCADuration	Optional	Set a value to identify signing certificates that might expire soon with a different value from other certificates. If the parameter value is not specified, the CA certificate expiration is the value used for the **minimumDuration**. See Golang Parse Duration for more information.
spec.maximumDuration	Optional	Set a value to identify certificates that have been created with a duration that exceeds your desired limit. The parameter uses the time duration format from Golang. See Golang Parse Duration for more information.

Field	Optional or required	Description
spec.maximumCADuration	Optional	Set a value to identify signing certificates that have been created with a duration that exceeds your defined limit. The parameter uses the time duration format from Golang. See Golang Parse Duration for more information.
spec.allowedSANPattern	Optional	A regular expression that must match every SAN entry that you have defined in your certificates. This parameter checks DNS names against patterns. See the Golang Regular Expression syntax for more information.
spec.disallowedSANPattern	Optional	A regular expression that must not match any SAN entries you have defined in your certificates. This parameter checks DNS names against patterns. **Note:** To detect wild-card certificate, use the following SAN pattern: **disallowedSANPattern: "[*]"** See the Golang Regular Expression syntax for more information.

1.1.2.2. Certificate policy sample

When your certificate policy controller is created on your hub cluster, a replicated policy is created on your managed cluster. See **policy-certificate.yaml** to view the certificate policy sample.

1.1.2.3. Additional resources

- Learn how to manage a certificate policy, see Managing security policies for more details.
- Refer to Policy controllers introduction for more topics.
- Return to the Certificates introduction.

1.1.3. Policy set controller

The policy set controller aggregates the policy status scoped to policies that are defined in the same namespace. Create a policy set (**PolicySet**) to group policies that are in the same namespace. All policies in the **PolicySet** are placed together in a selected cluster by creating a **PlacementBinding** to bind the **PolicySet** and **Placement**. The policy set is deployed to the hub cluster.

Additionally, when a policy is a part of multiple policy sets, existing and new **Placement** resources remain in the policy. When a user removes a policy from the policy set, the policy is not applied to the cluster that is selected in the policy set, but the placements remain. The policy set controller only checks for violations in clusters that include the policy set placement.

Notes:

- The Red Hat Advanced Cluster Management sample policy set uses cluster placement. If you use cluster placement, bind the namespace containing the policy to the managed cluster set. See Deploying policies to your cluster for more details on using cluster placement.
- In order to use a **Placement** resource, a **ManagedClusterSet** resource must be bound to the namespace of the **Placement** resource with a **ManagedClusterSetBinding** resource. Refer to Creating a *ManagedClusterSetBinding* resource for additional details.

Learn more details about the policy set structure in the following sections:

- Policy set controller YAML structure
- Policy set controller YAML table
- Policy set sample

1.1.3.1. Policy set YAML structure

Your policy set might resemble the following YAML file:

```
apiVersion: policy.open-cluster-management.io/v1beta1
kind: PolicySet
metadata:
  name: demo-policyset
spec:
  policies:
  - policy-demo

---
apiVersion: policy.open-cluster-management.io/v1
kind: PlacementBinding
metadata:
  name: demo-policyset-pb
placementRef:
  apiGroup: cluster.open-cluster-management.io
  kind: Placement
  name: demo-policyset-pr
subjects:
- apiGroup: policy.open-cluster-management.io
  kind: PolicySet
  name: demo-policyset
---
apiVersion: cluster.open-cluster-management.io/v1beta1
kind: Placement
metadata:
  name: demo-policyset-pr
spec:
  predicates:
  - requiredClusterSelector:
```

```
    labelSelector:
      matchExpressions:
        - key: name
          operator: In
          values:
            - local-cluster
  tolerations:
    - key: cluster.open-cluster-management.io/unavailable
      operator: Exists
    - key: cluster.open-cluster-management.io/unreachable
      operator: Exists
```

1.1.3.2. Policy set table

View the following parameter table for descriptions:

Table 1.3. Parameter table

Field	Optional or required	Description
apiVersion	Required	Set the value to **policy.open-cluster-management.io/v1beta1**.
kind	Required	Set the value to **PolicySet** to indicate the type of policy.
metadata.name	Required	The name for identifying the policy resource.
spec	Required	Add configuration details for your policy.
spec.policies	Optional	The list of policies that you want to group together in the policy set.

1.1.3.3. Policy set sample

```
apiVersion: policy.open-cluster-management.io/v1beta1
kind: PolicySet
metadata:
  name: pci
  namespace: default
spec:
  description: Policies for PCI compliance
  policies:
  - policy-pod
  - policy-namespace
status:
  compliant: NonCompliant
  placement:
```

```
- placementBinding: binding1
  placement: placement1
  policySet: policyset-ps
```

1.1.3.4. Additional resources

- See Red Hat OpenShift Platform Plus policy set .
- See the *Creating policy sets* section in the Managing security policies topic.
- Also view the stable **PolicySets**, which require the Policy Generator for deployment, PolicySets-- Stable.

1.1.4. Operator policy controller

The operator policy controller allows you to monitor and install Operator Lifecycle Manager operators across your clusters. Use the operator policy controller to monitor the health of various pieces of the operator and to specify how you want to automatically handle updates to the operator.

You can also distribute an operator policy to managed clusters by using the governance framework and adding the policy to the **policy-templates** field of a policy on the hub cluster.

You can also use template values within the **operatorGroup** and **subscription** fields of an operator policy. For more information, see *Template processing*.

1.1.4.1. Prerequisites

- Operator Lifecycle Manager must be available on your managed cluster. This is enabled by default on Red Hat OpenShift Container Platform.
- **Required access:** Cluster administrator

1.1.4.2. Operator policy YAML table

Field	Optional or required	Description
apiVersion	Required	Set the value to **policy.open-cluster-management.io/v1beta1**.
kind	Required	Set the value to **OperatorPolicy** to indicate the type of policy.
metadata.name	Required	The name for identifying the policy resource.

Field	Optional or required	Description
spec.remediationAction	Required	If the **remediationAction** for the operator policy is set to **enforce**, the controller creates resources on the target managed cluster to communicate to OLM to install the operator and approve updates based on the versions specified in the policy. + If the **remediationAction** set to **inform**, the controller only reports the status of the operator, including if any upgrades are available.
spec.operatorGroup	Optional	By default, if the **operatorGroup** field is not specified, the controller generates an **AllNamespaces** type **OperatorGroup** resource in the same namespace as the subscription, if supported. The operator policy controller produces this resource. View the following parameter descriptions: • **name**: The name of the **OperatorGroup** resource. • **namespace**: The namespace of the **OperatorGroup** resource. This parameter must match the namespace where you install the operator. • **targetNamespaces**: A list of namespaces that the operator is configured to watch and operate.
spec.complianceType	Required	Specifies the desired state of the operator on the cluster. If set to **musthave**, the policy is compliant when the operator is found. If set to **mustnothave**, the policy is compliant when the operator is not found.

Field	Optional or required	Description
spec.removalBehavior	Optional	Determines which resource types need to be kept or removed when you enforce an **OperatorPolicy** resource with **complianceType: mustnothave** defined. There is no effect when **complianceType** is set to **musthave**. View the following parameter descriptions: • **operatorGroups** can be set to **Keep** or **DeleteIfUnused**. The default value is **DeleteIfUnusued** which only removes the **OperatorGroup** resource if it is not used by any other operators. • **subscriptions** can be set to **Keep** or **Delete**. The default value is **Delete**. • **clusterServiceVersions** can be set to **Keep** or **Delete**. The default value is **Delete**. • **customResourceDefinitions** can be set to **Keep** or **Delete**. The default value is **Keep**. If you set this to **Delete**, the **CustomResourceDefintion** resources on the managed cluster are removed and can cause data loss.

Field	Optional or required	Description
spec.subscription	Required	Define the configurations to create an operator subscription. Add information in the following fields to create an operator subscription. View the following parameter descriptions: • **channel**: If not specified, the default channel is selected from the operator catalog. The default can be different on different OpenShift Container Platform versions. • **name**: Specify the package name of the operator. • **namespace**: If not specified, the default namespaced that is used for OpenShift Container Platform managed clusters is **openshift-operators**. • **source**: If not specified, the catalog that contains the operator is chosen. • **sourceNamespace**: If not specified, the namespace of the catalog that contains the operator is chosen. • **config**: Optional. Specify extra configuration details for the operator subscription. The **name** field is the only required field. The other fields are populated based on the catalog information within the managed cluster. **Note:** If you define a value for the **installPlanApproval** field in this parameter section, the policy becomes non-compliant. Use **spec.upgradeApproval** in the **OperatorPolicy** resource.

Field	Optional or required	Description
spec.complianceConfig	Optional	Use this parameter to define the compliance behavior for specific scenarios that are associated with operators. You can set each of the following options individually, where the supported values are **Compliant** and **NonCompliant**: • **catalogSourceUnhealthy**: Define the compliance when the catalog source for the operator is unhealthy. The default value is **Compliant** because this only affects possible upgrades. • **deploymentsUnavailable**: Define the compliance when at least one deployment of the operator is not fully available. The default value is **NonCompliant** because this reflects the availability of the service that the operator provides. • **upgradesAvailable**: Define the compliance when there is an upgrade available for the operator. The default value is **Compliant** because the existing operator installation might be running correctly.
spec.upgradeApproval	Required	If the **upgradeApproval** field is set to **Automatic**, version upgrades on the cluster are approved by the policy when the policy is set to **enforce**. If you set the field to **None**, version upgrades to the specific operator are not approved when the policy is set to **enforce**.

Field	Optional or required	Description
spec.versions	Optional	Declare which versions of the operator are compliant. If the field is empty, any version running on the cluster is considered compliant. If the field is not empty, the version on the managed cluster must match one of the versions in the list for the policy to be compliant. If the policy is set to **enforce** and the list is not empty, the versions listed here are approved by the controller on the cluster.

1.1.4.3. Additional resources

- See Template processing.
- See Installing an operator by using the **OperatorPolicy** resource for more details.
- See Managing operator policies in disconnected environments .
- See the Subscription topic in the OpenShift Container Platform documentation.
- See Operator Lifecycle Manager (OLM) for more details.
- See the Adding Operators to a cluster documentation for general information on OLM.

1.2. TEMPLATE PROCESSING

Configuration policies and operator policies support the inclusion of Golang text templates. These templates are resolved at runtime either on the hub cluster or the target managed cluster using configurations related to that cluster. This gives you the ability to define policies with dynamic content, and inform or enforce Kubernetes resources that are customized to the target cluster.

A policy definition can contain both hub cluster and managed cluster templates. Hub cluster templates are processed first on the hub cluster, then the policy definition with resolved hub cluster templates is propagated to the target clusters. A controller on the managed cluster processes any managed cluster templates in the policy definition and then enforces or verifies the fully resolved object definition.

The template must conform to the Golang template language specification, and the resource definition generated from the resolved template must be a valid YAML. See the Golang documentation about *Package templates* for more information. Any errors in template validation are recognized as policy violations. When you use a custom template function, the values are replaced at runtime.

Important:

- If you use hub cluster templates to propagate secrets or other sensitive data, the sensitive data exists in the managed cluster namespace on the hub cluster and on the managed clusters where that policy is distributed. The template content is expanded in the policy, and policies are not

encrypted by the OpenShift Container Platform ETCD encryption support. To address this, use **fromSecret** or **copySecretData**, which automatically encrypts the values from the secret, or **protect** to encrypt other values.

- When you add multiline string values such as, certificates, always add **| toRawJson | toLiteral** syntax at the end of the template pipeline to handle line breaks. For example, if you are copying a certificate from a **Secret** resource and including it in a **ConfigMap** resource, your template pipeline might be similar to the following syntax:

```
ca.crt: '{{ fromSecret "openshift-config" "ca-config-map-secret" "ca.crt"  | base64dec |
toRawJson | toLiteral }}'
```

The **toRawJson** template function converts the input value to a JSON string with new lines escaped to not affect the YAML structure. The **toLiteral** template function removes the outer single quotes from the output. For example, when templates are processed for the **key: '{{ 'hello\nworld' | toRawJson }}'** template pipeline, the output is **key: '"hello\nworld"'**. The output of the **key: '{{ 'hello\nworld' | toRawJson | toLiteral }}'** template pipeline is **key: "hello\nworld"**.

See the following table for a comparison of hub cluster and managed cluster templates:

1.2.1. Comparison of hub cluster and managed cluster templates

Table 1.4. Comparison table

Templates	Hub cluster	Managed cluster
Syntax	Golang text template specification	Golang text template specification
Delimiter	{{hub ... hub}}	{{ ... }}
Context	A **.ManagedClusterName** variable is available, which at runtime, resolves to the name of the target cluster where the policy is propagated. The **.ManagedClusterLabels** variable is also available, which resolves to a map of keys and values of the labels on the managed cluster where the policy is propagated.	No context variables

Templates	Hub cluster	Managed cluster
Access control	By default, you can only reference namespaced Kubernetes resources that are in the same namespace as the **Policy** object and the **ManagedCluster** object of the cluster that the policy propogates to. Alternatively, you can specify the **spec.hubTemplateOptions.serviceAccountName** field in the **Policy** object to a service account in the same namespace as the **Policy** resource. When you specify the field, the service account is used for all hub cluster template lookups. **Note:** The service account must have **list** and **watch** permissions on any resource that is looked up in a hub cluster template.	You can reference any resource on the cluster.
Functions	A set of template functions that support dynamic access to Kubernetes resources and string manipulation. See *Template functions* for more information. See the Access control row for lookup restrictions. The **fromSecret** template function on the hub cluster stores the resulting value as an encrypted string on the replicated policy, in the managed cluster namespace. The equivalent call might use the following syntax: **{{hub "(lookup "v1" "Secret" "default" "my-hub-secret").data.message | protect hub}}**	A set of template functions support dynamic access to Kubernetes resources and string manipulation. See *Template functions* for more information.

Templates	Hub cluster	Managed cluster
Function output storage	The output of template functions are stored in **Policy** resource objects in each applicable managed cluster namespace on the hub cluster, before it is synced to the managed cluster. This means that any sensitive results from template functions are readable by anyone with read access to the **Policy** resource objects on the hub cluster, and anyone with read access to the **ConfigurationPolicy** or **OperatorPolicy** resource objects on the managed clusters. Additionally, if etcd encryption is enabled, the policy resource objects are not encrypted. It is best to carefully consider this when using template functions that return sensitive output (e.g. from a secret).	The output of template functions are not stored in policy related resource objects.
Policy metadata	The **.PolicyMetadata** variable resolves to a map with the **name**, **namespace**, **labels**, and **annotations** keys with values from the root policy.	No context variables
Processing	Processing occurs at runtime on the hub cluster during propagation of replicated policies to clusters. Policies and the hub cluster templates within the policies are processed on the hub cluster only when templates are created or updated.	Processing occurs on the managed cluster. Configuration policies are processed periodically, which automatically updates the resolved object definition with data in the referenced resources. Operator policies automatically update whenever a referenced resource changes.
Processing errors	Errors from the hub cluster templates are displayed as violations on the managed clusters the policy applies to.	Errors from the managed cluster templates are displayed as violations on the specific target cluster where the violation occurred.

Continue reading the following topics:

- Template functions
- Advanced template processing in configuration policies

1.2.2. Template functions

Reference Kubernetes resources such as resource-specific and generic template functions on your hub cluster by using the **{{hub ... hub}}** delimiters, or on your managed cluster by using the **{{ ... }}** delimiters. You can use resource-specific functions for convenience and to make the content of your resources more accessible.

1.2.2.1. Template function descriptions

If you use the generic function, **lookup**, which is more advanced, familiarize yourself with the YAML structure of the resource that is being looked up. In addition to these functions, utility functions such as **base64enc**, **base64dec**, **indent**, **autoindent**, **toInt**, **toBool**, and more are available.

To conform templates with YAML syntax, you must define templates in the policy resource as strings using quotes or a block character (| or >). This causes the resolved template value to also be a string. To override this, use **toInt** or **toBool** as the final function in the template to initiate further processing that forces the value to be interpreted as an integer or boolean respectively.

Continue reading to view the descriptions and examples for some of the custom template functions that are supported:

- *fromSecret*
- *fromConfigMap*
- *fromClusterClaim*
- *lookup*
- *base64enc*
- *base64dec*
- *indent*
- *autoindent*
- *toInt*
- *toBool*
- *protect*
- *toLiteral*
- *copySecretData*
- *copyConfigMapData*
- *getNodesWithExactRoles*
- *hasNodesWithExactRoles*

- Sprig open source

1.2.2.1.1. *fromSecret*

The **fromSecret** function returns the value of the given data key in the secret. View the following syntax for the function:

```
func fromSecret (ns string, secretName string, datakey string) (dataValue string, err error)
```

When you use this function, enter the namespace, name, and data key of a Kubernetes **Secret** resource. You must use the same namespace that is used for the policy when using the function in a hub cluster template. See *Template processing* for more details.

View the following configuration policy that enforces a **Secret** resource on the target cluster:

```
apiVersion: policy.open-cluster-management.io/v1
kind: ConfigurationPolicy
metadata:
  name: demo-fromsecret
  namespace: test
spec:
  namespaceSelector:
    exclude:
    - kube-*
    include:
    - default
  object-templates:
  - complianceType: musthave
    objectDefinition:
      apiVersion: v1
      data: (1)
        USER_NAME: YWRtaW4=
        PASSWORD: '{{ fromSecret "default" "localsecret" "PASSWORD" }}' (2)
      kind: Secret (3)
      metadata:
        name: demosecret
        namespace: test
      type: Opaque
  remediationAction: enforce
  severity: low
```

(1) When you use this function with hub cluster templates, the output is automatically encrypted using the protect function.

(2) The value for the **PASSWORD** data key is a template that references the secret on the target cluster.

(3) You receive a policy violation if the Kubernetes **Secret** resource does not exist on the target cluster. If the data key does not exist on the target cluster, the value becomes an empty string.

Important: When you add multiline string values such as, certificates, always add | **toRawJson** | **toLiteral** syntax at the end of the template pipeline to handle line breaks. For example, if you are copying a certificate from a **Secret** resource and including it in a **ConfigMap** resource, your template pipeline might be similar to the following syntax:

```
ca.crt: '{{ fromSecret "openshift-config" "ca-config-map-secret" "ca.crt"  | base64dec | toRawJson |
toLiteral }}'
```

- The **toRawJson** template function converts the input value to a JSON string with new lines escaped to not affect the YAML structure.
- The **toLiteral** template function removes the outer single quotes from the output. For example, when templates are processed for the **key: '{{ 'hello\nworld' | toRawJson }}'** template pipeline, the output is **key: '"hello\nworld"'**. The output of the **key: '{{ 'hello\nworld' | toRawJson | toLiteral }}'** template pipeline is **key: "hello\nworld"**.

1.2.2.1.2. *fromConfigmap*

The **fromConfigMap** function returns the value of the given data key in the config map. When you use this function, enter the namespace, name, and data key of a Kubernetes **ConfigMap** resource. You must use the same namespace that is used for the policy using the function in a hub cluster template. See *Template processing* for more details.

View the following syntax for the function:

```
func fromConfigMap (ns string, configmapName string, datakey string) (dataValue string, err Error)
```

View the following configuration policy that enforces a Kubernetes resource on the target managed cluster:

```
apiVersion: policy.open-cluster-management.io/v1
kind: ConfigurationPolicy
metadata:
  name: demo-fromcm-lookup
  namespace: test-templates
spec:
  namespaceSelector:
    exclude:
    - kube-*
    include:
    - default
  object-templates:
  - complianceType: musthave
    objectDefinition:
      kind: ConfigMap (1)
      apiVersion: v1
      metadata:
        name: demo-app-config
        namespace: test
      data: (2)
        app-name: sampleApp
        app-description: "this is a sample app"
        log-file: '{{ fromConfigMap "default" "logs-config" "log-file" }}' (3)
        log-level: '{{ fromConfigMap "default" "logs-config" "log-level" }}' (4)
  remediationAction: enforce
  severity: low
```

(1) You receive a policy violation if the Kubernetes **ConfigMap** resource does not exist on the target cluster.

2 If the **data** key does not exist on the target cluster, the value becomes an empty string.

3 The value for the **log-file** data key is a template that retrieves the value of the **log-file** from the **logs-config** config map in the **default** namespace.

4 The **log-level** is a tempalte that retrieves the value of the **log-level** data key in the **default** namespace.

1.2.2.1.3. *fromClusterClaim*

The **fromClusterClaim** function returns the value of the **Spec.Value** in the **ClusterClaim** resource. View the following syntax for the function:

```
func fromClusterClaim (clusterclaimName string) (dataValue string, err Error)
```

View the following example of the configuration policy that enforces a Kubernetes resource on the target managed cluster:

```
apiVersion: policy.open-cluster-management.io/v1
kind: ConfigurationPolicy
metadata:
  name: demo-clusterclaims 1
  namespace: default
spec:
  namespaceSelector:
    exclude:
    - kube-*
    include:
    - default
  object-templates:
  - complianceType: musthave
    objectDefinition:
      kind: ConfigMap
      apiVersion: v1
      metadata:
        name: sample-app-config
        namespace: default
      data: 2
        platform: '{{ fromClusterClaim "platform.open-cluster-management.io" }}' 3
        product: '{{ fromClusterClaim "product.open-cluster-management.io" }}'
        version: '{{ fromClusterClaim "version.openshift.io" }}'
  remediationAction: enforce
  severity: low
```

1 When you use this function, enter the name of a Kubernetes **ClusterClaim** resource. You receive a policy violation if the **ClusterClaim** resource does not exist.

2 Configuration values can be set as key-value properties.

3 The value for the **platform** data key is a template that retrieves the value of the **platform.open-cluster-management.io** cluster claim. Similarly, it retrieves values for **product** and **version** from the **ClusterClaim** resource.

1.2.2.1.4. *lookup*

The **lookup** function returns the Kubernetes resource as a JSON compatible map. When you use this function, enter the API version, kind, namespace, name, and optional label selectors of the Kubernetes resource. You must use the same namespace that is used for the policy within the hub cluster template. See *Template processing* for more details.

If the requested resource does not exist, an empty map is returned. If the resource does not exist and the value is provided to another template function, you might get the following error: **invalid value; expected string**.

Note: Use the **default** template function, so the correct type is provided to later template functions. See the *Sprig open source* section.

View the following syntax for the function:

```
func lookup (apiversion string, kind string, namespace string, name string, labelselector ...string)
(value string, err Error)
```

For label selector examples, see the reference to the *Kubernetes labels and selectors* documentation, in the *Additional resources* section. View the following example of the configuration policy that enforces a Kubernetes resource on the target managed cluster:

```
apiVersion: policy.open-cluster-management.io/v1
kind: ConfigurationPolicy
metadata:
  name: demo-lookup
  namespace: test-templates
spec:
  namespaceSelector:
    exclude:
    - kube-*
    include:
    - default
  object-templates:
  - complianceType: musthave
    objectDefinition:
      kind: ConfigMap
      apiVersion: v1
      metadata:
        name: demo-app-config
        namespace: test
      data: (1)
        app-name: sampleApp
        app-description: "this is a sample app"
        metrics-url: | (2)
          http://{{ (lookup "v1" "Service" "default" "metrics").spec.clusterIP }}:8080
  remediationAction: enforce
  severity: low
```

(1) Configuration values can be set as key-value properties.

(2) The value for the **metrics-url** data key is a template that retrieves the **v1**/**Service** Kubernetes resource **metrics** from the **default** namespace, and is set to the value of the **Spec.ClusterIP** in the queried resource.

1.2.2.1.5. *base64enc*

The **base64enc** function returns a **base64** encoded value of the input **data string**. When you use this function, enter a string value. View the following syntax for the function:

```
func base64enc (data string) (enc-data string)
```

View the following example of the configuration policy that uses the **base64enc** function:

```
apiVersion: policy.open-cluster-management.io/v1
kind: ConfigurationPolicy
metadata:
  name: demo-fromsecret
  namespace: test
spec:
  namespaceSelector:
    exclude:
    - kube-*
    include:
    - default
  object-templates:
  - complianceType: musthave
    objectDefinition:
    ...
    data:
      USER_NAME: '{{ fromConfigMap "default" "myconfigmap" "admin-user" | base64enc }}'
```

1.2.2.1.6. *base64dec*

The **base64dec** function returns a **base64** decoded value of the input **enc-data string**. When you use this function, enter a string value. View the following syntax for the function:

```
func base64dec (enc-data string) (data string)
```

View the following example of the configuration policy that uses the **base64dec** function:

```
apiVersion: policy.open-cluster-management.io/v1
kind: ConfigurationPolicy
metadata:
  name: demo-fromsecret
  namespace: test
spec:
  namespaceSelector:
    exclude:
    - kube-*
    include:
    - default
  object-templates:
  - complianceType: musthave
    objectDefinition:
    ...
    data:
      app-name: |
        "{{ ( lookup "v1"  "Secret" "testns" "mytestsecret") .data.appname ) | base64dec }}"
```

1.2.2.1.7. *indent*

The **indent** function returns the padded **data string**. When you use this function, enter a data string with the specific number of spaces. View the following syntax for the function:

```
func indent (spaces  int,  data string) (padded-data string)
```

View the following example of the configuration policy that uses the **indent** function:

```
apiVersion: policy.open-cluster-management.io/v1
kind: ConfigurationPolicy
metadata:
  name: demo-fromsecret
  namespace: test
spec:
  namespaceSelector:
    exclude:
    - kube-*
    include:
    - default
  object-templates:
  - complianceType: musthave
    objectDefinition:
    ...
    data:
      Ca-cert:  |
        {{ ( index ( lookup "v1" "Secret" "default" "mycert-tls"  ).data  "ca.pem"  ) |  base64dec | indent 4
}}
```

1.2.2.1.8. *autoindent*

The **autoindent** function acts like the **indent** function that automatically determines the number of leading spaces based on the number of spaces before the template.

View the following example of the configuration policy that uses the **autoindent** function:

```
apiVersion: policy.open-cluster-management.io/v1
kind: ConfigurationPolicy
metadata:
  name: demo-fromsecret
  namespace: test
spec:
  namespaceSelector:
    exclude:
    - kube-*
    include:
    - default
  object-templates:
  - complianceType: musthave
    objectDefinition:
    ...
    data:
      Ca-cert:  |
        {{ ( index ( lookup "v1" "Secret" "default" "mycert-tls"  ).data  "ca.pem"  ) |  base64dec |
autoindent }}
```

-

1.2.2.1.9. *toInt*

The **toInt** function casts and returns the integer value of the input value. When this is the last function in the template, there is further processing of the source content. This is to ensure that the value is interpreted as an integer by the YAML. When you use this function, enter the data that needs to be casted as an integer. View the following syntax for the function:

```
func toInt (input interface{}) (output int)
```

View the following example of the configuration policy that uses the **toInt** function:

```
apiVersion: policy.open-cluster-management.io/v1
kind: ConfigurationPolicy
metadata:
  name: demo-template-function
  namespace: test
spec:
  namespaceSelector:
    exclude:
    - kube-*
    include:
    - default
  object-templates:
  - complianceType: musthave
    objectDefinition:
    ...
    spec:
      vlanid:  |
        {{ (fromConfigMap "site-config" "site1" "vlan")  | toInt }}
```

1.2.2.1.10. *toBool*

The **toBool** function converts the input string into a boolean, and returns the boolean. When this is the last function in the template, there is further processing of the source content. This is to ensure that the value is interpreted as a boolean by the YAML. When you use this function, enter the string data that needs to be converted to a boolean. View the following syntax for the function:

```
func toBool (input string) (output bool)
```

View the following example of the configuration policy that uses the **toBool** function:

```
apiVersion: policy.open-cluster-management.io/v1
kind: ConfigurationPolicy
metadata:
  name: demo-template-function
  namespace: test
spec:
  namespaceSelector:
    exclude:
    - kube-*
    include:
    - default
  object-templates:
```

```
  - complianceType: musthave
    objectDefinition:
    ...
    spec:
      enabled: |
        {{ (fromConfigMap "site-config" "site1" "enabled")  | toBool }}
```

1.2.2.1.11. *protect*

The **protect** function enables you to encrypt a string in a hub cluster policy template. It is automatically decrypted on the managed cluster when the policy is evaluated. View the following example of the configuration policy that uses the **protect** function:

```
apiVersion: policy.open-cluster-management.io/v1
kind: ConfigurationPolicy
metadata:
  name: demo-template-function
  namespace: test
spec:
  namespaceSelector:
    exclude:
    - kube-*
    include:
    - default
  object-templates:
  - complianceType: musthave
    objectDefinition:
    ...
    spec:
      enabled: |
        {{hub (lookup "v1" "Secret" "default" "my-hub-secret").data.message | protect hub}}
```

In the previous YAML example, there is an existing hub cluster policy template that is defined to use the **lookup** function. On the replicated policy in the managed cluster namespace, the value might resemble the following syntax: **$ocm_encrypted:okrrBqt72ol+3WT/0vxel3vGa+wpLD7Z0ZxFMLvL204=**

Each encryption algorithm used is AES-CBC using 256-bit keys. Each encryption key is unique per managed cluster and is automatically rotated every 30 days.

This ensures that your decrypted value is to never be stored in the policy on the managed cluster.

To force an immediate rotation, delete the **policy.open-cluster-management.io/last-rotated** annotation on the **policy-encryption-key** Secret in the managed cluster namespace on the hub cluster. Policies are then reprocessed to use the new encryption key.

1.2.2.1.12. *toLiteral*

The **toLiteral** function removes any quotation marks around the template string after it is processed. You can use this function to convert a JSON string from a config map field to a JSON value in the manifest. Run the following function to remove quotation marks from the **key** parameter value:

```
key: '{{ "[\"10.10.10.10\", \"1.1.1.1\"]" | toLiteral }}'
```

After using the **toLiteral** function, the following update is displayed:

```
key: ["10.10.10.10", "1.1.1.1"]
```

1.2.2.1.13. *copySecretData*

The **copySecretData** function copies all of the **data** contents of the specified secret. View the following sample of the function:

```
complianceType: musthave
    objectDefinition:
      apiVersion: v1
      kind: Secret
      metadata:
        name: my-secret-copy
      data: '{{ copySecretData "default" "my-secret" }}' 1
```

1 When you use this function with hub cluster templates, the output is automatically encrypted using the protect function.

1.2.2.1.14. *copyConfigMapData*

The **copyConfigMapData** function copies all of the **data** content of the specified config map. View the following sample of the function:

```
complianceType: musthave
    objectDefinition:
      apiVersion: v1
      kind: ConfigMap
      metadata:
        name: my-secret-copy
      data: '{{ copyConfigMapData "default" "my-configmap" }}'
```

1.2.2.1.15. *getNodesWithExactRoles*

The **getNodesWithExactRoles** function returns a list of nodes with only the roles that you specify, and ignores nodes that have any additional roles except the **node-role.kubernetes.io/worker** role. View the following sample function where you are selecting **"infra"** nodes and ignoring the storage nodes:

```
    complianceType: musthave
    objectDefinition:
      apiVersion: v1
      kind: ConfigMap
      metadata:
        name: my-configmap
        data:
          infraNode: |
            {{- range $i,$nd := (getNodesWithExactRoles "infra").items }}
            node{{ $i }}: {{ $nd.metadata.name }}
            {{- end }}
          replicas: {{ len ((getNodesWithExactRoles "infra").items) | toInt }}
```

1.2.2.1.16. *hasNodesWithExactRoles*

The **hasNodesWithExactRoles** function returns the **true** value if the cluster contains nodes with only the roles that you specify, and ignores nodes that have any additional roles except the **node-role.kubernetes.io/worker** role. View the following sample of the function:

```
complianceType: musthave
objectDefinition:
  apiVersion: v1
  kind: ConfigMap
  metadata:
    name: my-configmap
  data:
    key: '{{ hasNodesWithExactRoles "infra" }}'
```

1.2.2.1.17. Sprig open source

Red Hat Advanced Cluster Management supports the following template functions that are included from the **sprig** open source project:

Table 1.5. Table of supported, community Sprig functions

Sprig library	Functions
Cryptographic and security	**htpasswd**
Date	**date**, **mustToDate**, **now**, **toDate**
Default	**default**, **empty**, **fromJson**, **mustFromJson**, **ternary**, **toJson**, **toRawJson**
Dictionaries and dict	**dict**, **dig**, **get**, **hasKey**, **merge**, **mustMerge**, **set**, **unset**
Integer math	**add**, **mul**, **div**, **round**, **sub**
Integer slice	**until**, **untilStep**,
Lists	**append**, **concat**, **has**, **list**, **mustAppend**, **mustHas**, **mustPrepend**, **mustSlice**, **prepend**, **slice**
String functions	**cat**, **contains**, **hasPrefix**, **hasSuffix**, **join**, **lower**, **mustRegexFind**, **mustRegexFindAll**, **mustRegexMatch**, **quote**, **regexFind**, **regexFindAll**, **regexMatch**, **regexQuoteMeta**, **replace**, **split**, **splitn**, **substr**, **trim**, **trimAll**, **trunc**, **upper**
Version comparison	**semver**, **semverCompare**

1.2.2.2. Additional resources

- See Template processing for more details.
- See Advanced template processing in configuration policies for use-cases.
- For label selector examples, see the Kubernetes labels and selectors documentation.
- Refer to the Golang documentation - Package templates.
- See the Sprig Function Documentation for more details.

1.2.3. Advanced template processing in configuration policies

Use both managed cluster and hub cluster templates to reduce the need to create separate policies for each target cluster or hardcode configuration values in the policy definitions. For security, both resource-specific and the generic lookup functions in hub cluster templates are restricted to the namespace of the policy on the hub cluster.

Important: If you use hub cluster templates to propagate secrets or other sensitive data, that causes sensitive data exposure in the managed cluster namespace on the hub cluster and on the managed clusters where that policy is distributed. The template content is expanded in the policy, and policies are not encrypted by the OpenShift Container Platform ETCD encryption support. To address this, use **fromSecret** or **copySecretData**, which automatically encrypts the values from the secret, or **protect** to encrypt other values.

Continue reading for advanced template use-cases:

- Special annotation for reprocessing
- Object template processing
- Bypass template processing

1.2.3.1. Special annotation for reprocessing

Hub cluster templates are resolved to the data in the referenced resources during policy creation, or when the referenced resources are updated.

If you need to manually initiate an update, use the special annotation, **policy.open-cluster-management.io/trigger-update**, to indicate changes for the data referenced by the templates. Any change to the special annotation value automatically initiates template processing. Additionally, the latest contents of the referenced resource are read and updated in the policy definition that is propagated for processing on managed clusters. A way to use this annotation is to increment the value by one each time.

1.2.3.2. Object template processing

Set object templates with a YAML string representation. The **object-template-raw** parameter is an optional parameter that supports advanced templating use-cases, such as if-else and the **range** function. The following example is defined to add the **species-category: mammal** label to any ConfigMap in the **default** namespace that has a **name** key equal to **Sea Otter**:

```
object-templates-raw: |
  {{- range (lookup "v1" "ConfigMap" "default" "").items }}
```

```
  {{- if eq .data.name "Sea Otter" }}
  - complianceType: musthave
    objectDefinition:
      kind: ConfigMap
      apiVersion: v1
      metadata:
        name: {{ .metadata.name }}
        namespace: {{ .metadata.namespace }}
        labels:
          species-category: mammal
  {{- end }}
  {{- end }}
```

Note: While **spec.object-templates** and **spec.object-templates-raw** are optional, exactly one of the two parameter fields must be set.

View the following policy example that uses advanced templates to create and configure infrastructure **MachineSet** objects for your managed clusters.

```
apiVersion: policy.open-cluster-management.io/v1
kind: ConfigurationPolicy
metadata:
  name: create-infra-machineset
spec:
  remediationAction: enforce
  severity: low
  object-templates-raw: |
    {{- /* Specify the parameters needed to create the MachineSet  */ -}}
    {{- $machineset_role := "infra" }}
    {{- $region := "ap-southeast-1" }}
    {{- $zones := list "ap-southeast-1a" "ap-southeast-1b" "ap-southeast-1c" }}
    {{- $infrastructure_id := (lookup "config.openshift.io/v1" "Infrastructure" ""
"cluster").status.infrastructureName }}
    {{- $worker_ms := (index (lookup "machine.openshift.io/v1beta1" "MachineSet" "openshift-
machine-api" "").items 0) }}
    {{- /* Generate the MachineSet for each zone as specified  */ -}}
    {{- range $zone := $zones }}
    - complianceType: musthave
      objectDefinition:
        apiVersion: machine.openshift.io/v1beta1
        kind: MachineSet
        metadata:
          labels:
            machine.openshift.io/cluster-api-cluster: {{ $infrastructure_id }}
          name: {{ $infrastructure_id }}-{{ $machineset_role }}-{{ $zone }}
          namespace: openshift-machine-api
        spec:
          replicas: 1
          selector:
            matchLabels:
              machine.openshift.io/cluster-api-cluster: {{ $infrastructure_id }}
              machine.openshift.io/cluster-api-machineset: {{ $infrastructure_id }}-{{ $machineset_role }}-{{
$zone }}
          template:
            metadata:
              labels:
```

```
            machine.openshift.io/cluster-api-cluster: {{ $infrastructure_id }}
            machine.openshift.io/cluster-api-machine-role: {{ $machineset_role }}
            machine.openshift.io/cluster-api-machine-type: {{ $machineset_role }}
            machine.openshift.io/cluster-api-machineset: {{ $infrastructure_id }}-{{ $machineset_role }}-
{{ $zone }}
        spec:
          metadata:
            labels:
              node-role.kubernetes.io/{{ $machineset_role }}: ""
          taints:
            - key: node-role.kubernetes.io/{{ $machineset_role }}
              effect: NoSchedule
          providerSpec:
            value:
              ami:
                id: {{ $worker_ms.spec.template.spec.providerSpec.value.ami.id }}
              apiVersion: awsproviderconfig.openshift.io/v1beta1
              blockDevices:
                - ebs:
                    encrypted: true
                    iops: 2000
                    kmsKey:
                      arn: ''
                    volumeSize: 500
                    volumeType: io1
              credentialsSecret:
                name: aws-cloud-credentials
              deviceIndex: 0
              instanceType: {{ $worker_ms.spec.template.spec.providerSpec.value.instanceType }}
              iamInstanceProfile:
                id: {{ $infrastructure_id }}-worker-profile
              kind: AWSMachineProviderConfig
              placement:
                availabilityZone: {{ $zone }}
                region: {{ $region }}
              securityGroups:
                - filters:
                    - name: tag:Name
                      values:
                        - {{ $infrastructure_id }}-worker-sg
              subnet:
                filters:
                  - name: tag:Name
                    values:
                      - {{ $infrastructure_id }}-private-{{ $zone }}
              tags:
                - name: kubernetes.io/cluster/{{ $infrastructure_id }}
                  value: owned
              userDataSecret:
                name: worker-user-data
  {{- end }}
```

1.2.3.3. Bypass template processing

You might create a policy that contains a template that is not intended to be processed by Red Hat Advanced Cluster Management. By default, Red Hat Advanced Cluster Management processes all templates.

To bypass template processing for your hub cluster, you must change **{{ template content }}** to **{{ `{{ template content }}` }}**.

Alternatively, you can add the following annotation in the **ConfigurationPolicy** section of your **Policy**: **policy.open-cluster-management.io/disable-templates: "true"**. When this annotation is included, the previous workaround is not necessary. Template processing is bypassed for the **ConfigurationPolicy**.

1.2.3.4. Additional resources

- See Template functions for more details.
- Return to Template processing.
- See Kubernetes configuration policy controller for more details.
- Also refer to the Backing up etcd data.

CHAPTER 2. POLICY DEPLOYMENT

The Kubernetes API is used to define and interact with policies. You are responsible for ensuring that the environment meets internal enterprise security standards for software engineering, secure engineering, resiliency, security, and regulatory compliance for workloads hosted on Kubernetes clusters.

2.1. DEPLOYMENT OPTIONS

The **CertificatePolicy**, **ConfigurationPolicy**, and **OperatorPolicy** policies and Gatekeeper constraints can be deployed to your managed clusters in two ways. You can deploy policies to your managed clusters by using the *Hub cluster policy framework* or *Deploying policies with external tools*. View the following descriptions of each option:

Hub cluster policy framework

The first approach is to leverage the policy framework on the hub cluster. This involves defining the policies and Gatekeeper constraints within a **Policy** object on the hub cluster and leveraging a **Placement** and **PlacementBinding** to select which clusters the policies get deployed to. The policy framework handles the delivery to the managed cluster and the status aggregation back to the hub cluster. The **Policy** objects are represented in the *Policies* table in the Red Hat Advanced Cluster Management for Kubernetes console.

Deploying policies with external tools

Alternatively, you can use an external tool, such as Red Hat OpenShift GitOps, to deliver the policies and Gatekeeper constraints directly to the managed cluster. Your policies are shown in the *Discovered policies* table in the Red Hat Advanced Cluster Management console and can be a helpful option if a configuration management strategy is already in place, but there is a desire to supplement the strategy with Red Hat Advanced Cluster Management policies.

2.1.1. Policy deployment comparison table

See the following comparison table to learn which option supports specific features, which helps you make a decision on which deployment strategy is more suitable for your use case:

Feature	Policy framework	Deployed with external tools
Hub cluster templating	Yes	No
Managed cluster templating	Yes	Yes
Red Hat Advanced Cluster Management console support	Yes	Yes, you can view policies that are deployed by external tools from the *Discover policies* table. Policies that are deployed with external tools are not displayed from the Governance *Overview* dashboard. You must enable Search on your managed cluster.

Feature	Policy framework	Deployed with external tools
Policy grouping	Yes, you can have a combination of statuses and deployments through **Policy** and **PolicySet** resources.	You cannot use policy grouping directly on your policies when deployed from external tools, but Argo CD **Application** objects for each grouping gives a high-level status.
Policy event history	You can view the last 10 events per cluster per policy stored on the hub cluster.	No, but you can scrape the policy event history from the controller logs on each managed cluster.
Policy dependencies	Yes	No. Alternatively, you can use the Argo CD sync waves feature to define dependencies.
Policy Generator	Yes	No
OpenShift GitOps health checks You must complete extra configuration for Argo CD versions earlier than 2.13.	Yes	Yes
Policy compliance history API (Technology Preview) (Deprecated)	Yes	No
OpenShift GitOps applying native Kubernetes manifests and Red Hat Advanced Cluster Management policy on the managed cluster	No, you must deploy a policy on your Red Hat Advanced Cluster Management hub cluster.	Yes
Policy status metric on the hub cluster for alerts	Yes	No
Running Ansible jobs on violated policies	Yes, use the **PolicyAutomation** resource.	No

2.2. ADDITIONAL RESOURCES

- Hub cluster policy framework
- Policy deployment with external tools

2.3. HUB CLUSTER POLICY FRAMEWORK

To create and manage policies, gain visibility and remediate configurations to meet standards, use the Red Hat Advanced Cluster Management for Kubernetes Governance security policy framework. Red Hat Advanced Cluster Management for Kubernetes governance provides an extensible policy framework for enterprises to introduce their own security policies.

You can create a **Policy** resource in any namespace on the hub cluster except the managed cluster namespaces. If you create a policy in a managed cluster namespace, it is deleted by Red Hat Advanced Cluster Management. Each Red Hat Advanced Cluster Management policy can be organized into one or more policy template definitions. For more details about the policy elements, view the *Policy YAML table* section on this page.

2.3.1. Requirements

- Each policy requires a **Placement** resource that defines the clusters that the policy document is applied to, and a **PlacementBinding** resource that binds the Red Hat Advanced Cluster Management for Kubernetes policy.
  Policy resources are applied to cluster based on an associated **Placement** definition, where you can view a list of managed clusters that match a certain criteria. A sample **Placement** resource that matches all clusters that have the **environment=dev** label resembles the following YAML:

```
apiVersion: cluster.open-cluster-management.io/v1beta1
kind: Placement
metadata:
  name: placement-policy-role
spec:
  predicates:
  - requiredClusterSelector:
    labelSelector:
    matchExpressions:
      - {key: environment, operator: In, values: ["dev"]}
    tolerations:
      - key: cluster.open-cluster-management.io/unavailable
        operator: Exists
      - key: cluster.open-cluster-management.io/unreachable
        operator: Exists
```

  To learn more about placement and the supported criteria, see Placement overview in the Cluster lifecycle documentation.

- In addition to the **Placement** resource, you must create a **PlacementBinding** to bind your **Placement** resource to a policy. A sample **Placement** resource that matches all clusters that have the **environment=dev** label resembles the following YAML:

```
apiVersion: policy.open-cluster-management.io/v1
kind: PlacementBinding
metadata:
  name: binding-policy-role
placementRef:
  name: placement-policy-role 1
  kind: Placement
  apiGroup: cluster.open-cluster-management.io
subjects: 2
```

```
- name: policy-role
  kind: Policy
  apiGroup: policy.open-cluster-management.io
```

1. If you use the previous sample, make sure you update the name of the **Placement** resource in the **placementRef** section to match your placement name.

2. You must update the name of the policy in the **subjects** section to match your policy name. Use the **oc apply -f resource.yaml -n namespace** command to apply both the **Placement** and the **Placementbinding** resources. Make sure the policy, placement and placement binding are all created in the same namespace.

- To use a **Placement** resource, you must bind a **ManagedClusterSet** resource to the namespace of the **Placement** resource with a **ManagedClusterSetBinding** resource. Refer to Creating a *ManagedClusterSetBinding* resource for additional details.

- When you create a **Placement** resource for your policy from the console, the status of the placement toleration is automatically added to the **Placement** resource. See Adding a toleration to a placement for more details.

Best practice: Use the command line interface (CLI) to make updates to the policies when you use the **Placement** resource.

2.3.2. Hub cluster policy components

Learn more details about the policy components in the following sections:

- Policy YAML structure
- Policy YAML table
- Policy sample file

2.3.2.1. Policy YAML structure

When you create a policy, you must include required parameter fields and values. Depending on your policy controller, you might need to include other optional fields and values. View the following YAML structure for policies:

```
apiVersion: policy.open-cluster-management.io/v1
kind: Policy
metadata:
  name:
  annotations:
    policy.open-cluster-management.io/standards:
    policy.open-cluster-management.io/categories:
    policy.open-cluster-management.io/controls:
    policy.open-cluster-management.io/description:
spec:
  disabled:
  remediationAction:
  dependencies:
  - apiVersion: policy.open-cluster-management.io/v1
    compliance:
```

```
    kind: Policy
    name:
    namespace:
  policy-templates:
    - objectDefinition:
        apiVersion:
        kind:
        metadata:
          name:
        spec:
---
apiVersion: policy.open-cluster-management.io/v1
kind: PlacementBinding
metadata:
bindingOverrides:
  remediationAction:
subFilter:
  name:
placementRef:
  name:
  kind: Placement
  apiGroup: cluster.open-cluster-management.io
subjects:
- name:
  kind:
  apiGroup:
---
apiVersion: cluster.open-cluster-management.io/v1beta1
kind: Placement
metadata:
  name:
spec:
```

2.3.2.2. Policy YAML table

View the following table for policy parameter descriptions:

Table 2.1. Parameter table

Field	Optional or required	Description
apiVersion	Required	Set the value to **policy.open-cluster-management.io/v1**.
kind	Required	Set the value to **Policy** to indicate the type of policy.
metadata.name	Required	The name for identifying the policy resource.

Field	Optional or required	Description
metadata.annotations	Optional	Used to specify a set of security details that describes the set of standards the policy is trying to validate. All annotations documented here are represented as a string that contains a comma-separated list. **Note:** You can view policy violations based on the standards and categories that you define for your policy on the *Policies* page, from the console.
bindingOverrides.remediationAction	Optional	When this parameter is set to **enforce**, it provides a way for you to override the remediation action of the related **PlacementBinding** resources for configuration policies. The default value is **null**.
subFilter	Optional	Set this parameter to **restriction** to select a subset of bound policies. The default value is **null**.
annotations.policy.open-cluster-management.io/standards	Optional	The name or names of security standards the policy is related to. For example, National Institute of Standards and Technology (NIST) and Payment Card Industry (PCI).
annotations.policy.open-cluster-management.io/categories	Optional	A security control category represent specific requirements for one or more standards. For example, a System and Information Integrity category might indicate that your policy contains a data transfer protocol to protect personal information, as required by the HIPAA and PCI standards.
annotations.policy.open-cluster-management.io/controls	Optional	The name of the security control that is being checked. For example, Access Control or System and Information Integrity.

Field	Optional or required	Description
spec.disabled	Required	Set the value to **true** or **false**. The **disabled** parameter provides the ability to enable and disable your policies.
spec.remediationAction	Optional	Specifies the remediation of your policy. The parameter values are **enforce** and **inform**. If specified, the **spec.remediationAction** value that is defined overrides any **remediationAction** parameter defined in the child policies in the **policy-templates** section. For example, if the **spec.remediationAction** value is set to **enforce**, then the **remediationAction** in the **policy-templates** section is set to **enforce** during runtime.
spec.copyPolicyMetadata	Optional	Specifies whether the labels and annotations of a policy should be copied when replicating the policy to a managed cluster. If you set to **true**, all of the labels and annotations of the policy are copied to the replicated policy. If you set to **false**, only the policy framework specific policy labels and annotations are copied to the replicated policy.
spec.dependencies	Optional	Used to create a list of dependency objects detailed with extra considerations for compliance.
spec.policy-templates	Required	Used to create one or more policies to apply to a managed cluster.
spec.policy-templates.extraDependencies	Optional	For policy templates, this is used to create a list of dependency objects detailed with extra considerations for compliance.

Field	Optional or required	Description
spec.policy-templates.ignorePending	Optional	Used to mark a policy template as compliant until the dependency criteria is verified. **Important:** Some policy kinds might not support the enforce feature.

2.3.2.3. Policy sample file

View the following YAML file which is a configuration policy for roles:

```
apiVersion: policy.open-cluster-management.io/v1
kind: Policy
metadata:
  name: policy-role
  annotations:
    policy.open-cluster-management.io/standards: NIST SP 800-53
    policy.open-cluster-management.io/categories: AC Access Control
    policy.open-cluster-management.io/controls: AC-3 Access Enforcement
    policy.open-cluster-management.io/description:
spec:
  remediationAction: inform
  disabled: false
  policy-templates:
    - objectDefinition:
        apiVersion: policy.open-cluster-management.io/v1
        kind: ConfigurationPolicy
        metadata:
          name: policy-role-example
        spec:
          remediationAction: inform # the policy-template spec.remediationAction is overridden by the preceding parameter value for spec.remediationAction.
          severity: high
          namespaceSelector:
            include: ["default"]
          object-templates:
            - complianceType: mustonlyhave # role definition should exact match
              objectDefinition:
                apiVersion: rbac.authorization.k8s.io/v1
                kind: Role
                metadata:
                  name: sample-role
                rules:
                  - apiGroups: ["extensions", "apps"]
                    resources: ["deployments"]
                    verbs: ["get", "list", "watch", "delete","patch"]
---
apiVersion: policy.open-cluster-management.io/v1
kind: PlacementBinding
metadata:
  name: binding-policy-role
```

```
placementRef:
  name: placement-policy-role
  kind: Placement
  apiGroup: cluster.open-cluster-management.io
subjects:
- name: policy-role
  kind: Policy
  apiGroup: policy.open-cluster-management.io
---
apiVersion: cluster.open-cluster-management.io/v1beta1
kind: Placement
metadata:
  name: placement-policy-role
spec:
  predicates:
  - requiredClusterSelector:
      labelSelector:
        matchExpressions:
        - {key: environment, operator: In, values: ["dev"]}
    tolerations:
      - key: cluster.open-cluster-management.io/unavailable
        operator: Exists
      - key: cluster.open-cluster-management.io/unreachable
        operator: Exists
```

2.3.3. Additional resources

Continue reading the related topics of the Red Hat Advanced Cluster Management governance framework:

- Governance policy framework architecture
- Policy controllers.
- Governance dashboard
- Managing security policies
- Policy dependencies
- Configuring policy compliance history (Technology Preview)
- Return to the Policy deployment

2.3.4. Policy dependencies

Dependencies can be used to activate a policy only when other policies on your cluster are in a certain state. When the dependency criteria is not met, the policy is labeled as **Pending** and resources are not created on your managed cluster. There are more details about the the criteria status in the policy status.

You can use policy dependencies to control the ordering of how objects are applied. For example, if you have a policy for an operator and another policy for a resource that the operator manages, you can set a dependency on the second policy so that it does not attempt to create the resource until the operator is installed. This can help with the performance on the managed cluster.

Required access: Policy administrator

View the following policy dependency example, where the **ScanSettingBinding** is only created if the **upstream-compliance-operator** policy is already compliant on the managed cluster:

```
apiVersion: policy.open-cluster-management.io/v1
kind: Policy
metadata:
  annotations:
    policy.open-cluster-management.io/categories: CM Configuration Management
    policy.open-cluster-management.io/controls: CM-2 Baseline Configuration
    policy.open-cluster-management.io/standards: NIST SP 800-53
    policy.open-cluster-management.io/description:
  name: moderate-compliance-scan
  namespace: default
spec:
  dependencies: 1
  - apiVersion: policy.open-cluster-management.io/v1
    compliance: Compliant
    kind: Policy
    name: upstream-compliance-operator
    namespace: default
  disabled: false
  policy-templates:
  - extraDependencies: 2
    - apiVersion: policy.open-cluster-management.io/v1
      kind: ConfigurationPolicy
      name: scan-setting-prerequisite
      compliance: Compliant
    ignorePending: false 3
    objectDefinition:
      apiVersion: policy.open-cluster-management.io/v1
      kind: ConfigurationPolicy
      metadata:
        name: moderate-compliance-scan
      spec:
        object-templates:
        - complianceType: musthave
          objectDefinition:
            apiVersion: compliance.openshift.io/v1alpha1
            kind: ScanSettingBinding
            metadata:
              name: moderate
              namespace: openshift-compliance
            profiles:
            - apiGroup: compliance.openshift.io/v1alpha1
              kind: Profile
              name: ocp4-moderate
            - apiGroup: compliance.openshift.io/v1alpha1
              kind: Profile
              name: ocp4-moderate-node
            settingsRef:
              apiGroup: compliance.openshift.io/v1alpha1
              kind: ScanSetting
```

```
      name: default
remediationAction: enforce
severity: low
```

1. The **dependencies** field is set on a **Policy** object, and the requirements apply to all policy templates in the policy.

2. The **extraDependencies** field can be set on individual policy template. For example the parameter can be set for a configuration policy, and defines criteria that must be satisfied in addition to any **dependencies** set in the policy.

3. The **ignorePending** field can be set on each individual policy template, and configures whether the **Pending** status on that template is considered as **Compliant** or **NonCompliant** when the overall policy compliance is calculated. By default, this is set to **false** and a **Pending** template causes the policy to be **NonCompliant**. When you set this to **true** the policy can still be **Compliant** when this template is **Pending**, which is useful when that is expected status of the template.

Note: You cannot use a dependency to apply a policy on one cluster based on the status of a policy in another cluster.

2.3.5. Configuring policy compliance history API (Technology Preview) (Deprecated)

The policy compliance history API is an optional technical preview feature if you want long-term storage of Red Hat Advanced Cluster Management for Kubernetes policy compliance events in a queryable format. You can use the API to get additional details such as the **spec** field to audit and troubleshoot your policy, and get compliance events when a policy is disabled or removed from a cluster.

The policy compliance history API can also generate a comma-separated values (CSV) spreadsheet of policy compliance events to help you with auditing and troubleshooting.

- Prerequisites
- Enabling the compliance history API
- Setting the compliance history API URL

2.3.5.1. Prerequisites

- The policy compliance history API requires a PostgreSQL server on version 13 or newer. Some Red Hat supported options include using the **registry.redhat.io/rhel9/postgresql-15** container image, the **registry.redhat.io/rhel8/postgresql-13** container image, the **postgresql-server** RPM, or **postgresql/server** module. Review the applicable official Red Hat documentation on setup and configuration for the path you choose. The policy compliance history API is compatible with any standard PostgreSQL and is not limited to the official Red Hat supported offerings.

- This PostgreSQL server must be reachable from the Red Hat Advanced Cluster Management hub cluster. If the PostgreSQL server is running externally of the hub cluster, ensure the routing and firewall configuration allows the hub cluster to connect to port 5432 of the PostgreSQL server. This port might be a different value if it is overridden in the PostgreSQL configuration.

2.3.5.2. Enabling the compliance history API

Configure your managed clusters to record policy compliance events to the API. You can enable this on all clusters or a subset of clusters. Complete the following steps:

1. Configure the PostgreSQL server as a cluster administrator. If you deployed PostgreSQL on your Red Hat Advanced Cluster Management hub cluster, temporarily port-forward the PostgreSQL port to use the **psql** command. Run the following command:

```
oc -n <PostgreSQL namespace> port-forward <PostgreSQL pod name> 5432:5432
```

2. In a different terminal, connect to the PostgreSQL server locally similar to the following command:

```
psql 'postgres://postgres:@127.0.0.1:5432/postgres'
```

3. Create a user and database for your Red Hat Advanced Cluster Management hub cluster with the following SQL statements:

```
CREATE USER "rhacm-policy-compliance-history" WITH PASSWORD '<replace with password>';
CREATE DATABASE "rhacm-policy-compliance-history" WITH OWNER="rhacm-policy-compliance-history";
```

4. Create the **governance-policy-database Secret** resource to use this database for the policy compliance history API. Run the following command:

```
oc -n open-cluster-management create secret generic governance-policy-database \ 1
  --from-literal="user=rhacm-policy-compliance-history" \
  --from-literal="password=rhacm-policy-compliance-history" \
  --from-literal="host=<replace with host name of the Postgres server>" \ 2
  --from-literal="dbname=ocm-compliance-history" \
 --from-literal="sslmode=verify-full" \
  --from-file="ca=<replace>" 3
```

1. Add the namespace where Red Hat Advanced Cluster Management is installed. By default, Red Hat Advanced Cluster Management is installed in the **open-cluster-management** namespace.

2. Add the host name of the PostgresQL server. If you deployed the PostgreSQL server on the Red Hat Advanced Cluster Management hub cluster and it is not exposed outside of the cluster, you can use the **Service** object for the host value. The format is **<service name>.<namespace>.svc**. Note, this approach depends on the network policies of the Red Hat Advanced Cluster Management hub cluster.

3. You must specify the Certificate Authority certificate file in the **ca** data field that signed the TLS certificate of the PostgreSQL server. If you do not provide this value, you must change the *sslmode* value accordingly, though it is not recommended since it reduces the security of the database connection.

5. Add the **cluster.open-cluster-management.io/backup** label to backup the **Secret** resource for a Red Hat Advanced Cluster Management hub cluster restore operation. Run the following command:

```
oc -n open-cluster-management label secret governance-policy-database cluster.open-cluster-management.io/backup=""
```

6. For more customization of the PostgreSQL connection, use the **connectionURL** data field directly and provide a value in the format of a PostgreSQL connection URI. Special characters in the password must be URL encoded. One option is to use Python to generate the URL encoded format of the password. For example, if the password is **$uper<Secr&t%>**, run the following Python command to get the output **%24uper%3CSecr%26t%25%3E**:

```
python -c 'import urllib.parse; import sys; print(urllib.parse.quote(sys.argv[1]))' '$uper<Secr&t%>'
```

7. Run the command to test the policy compliance history API after you create the **governance-policy-database Secret**. An OpenShift **Route** object is automatically created in the same namespace. If routes on the Red Hat Advanced Cluster Management hub cluster do not utilize a trusted certificate, you can choose to provide the **-k** flag in the curl command to skip TLS verification, though this is not recommended:

```
curl -H "Authorization: Bearer $(oc whoami --show-token)" \
   "https://$(oc -n open-cluster-management get route governance-history-api -o jsonpath='{.spec.host}')/api/v1/compliance-events"
```

   - If successful, the curl command returns a value similar to the following message:

```
{"data":[],"metadata":{"page":1,"pages":0,"per_page":20,"total":0}}
```

   - If it is not successful, the curl command might return either of the two messages:

```
{"message":"The database is unavailable"}
```

```
{"message":"Internal Error"}
```

   - If you receive a message, view the Kubernetes events in the **open-cluster-management** namespace with the following command:

```
oc -n open-cluster-management get events --field-selector reason=OCMComplianceEventsDBError
```

     a. If you receive instructions from the event to view the **governance-policy-propagator** logs, run the following command:

```
oc -n open-cluster-management logs -l name=governance-policy-propagator -f
```

     You might receive an error message that indicates the user, password, or database is incorrectly specified. See the following message example:

```
2024-03-05T12:17:14.500-0500 info compliance-events-api complianceeventsapi/complianceeventsapi_controller.go:261 The database connection failed: pq: password authentication failed for user "rhacm-policy-compliance-history"
```

8. Update the **governance-policy-database Secret** resource with the correct PostgreSQL connection settings with the following command:

```
oc -n open-cluster-management edit secret governance-policy-database
```

2.3.5.3. Setting the compliance history API URL

Set the policy compliance history API URL to enable the feature on managed clusters. Complete the following steps:

1. Retrieve the external URL of the policy compliance history API with the following command:

```
echo "https://$(oc -n open-cluster-management get route governance-history-api -o=jsonpath='{.spec.host}')"
```

The output might resemble the following information, with the domain name of your Red Hat Advanced Cluster Management hub cluster:

```
https://governance-history-api-open-cluster-management.apps.openshift.redhat.com
```

2. Create an **AddOnDeploymentConfig** object similar to the following example:

```
apiVersion: addon.open-cluster-management.io/v1alpha1
kind: AddOnDeploymentConfig
metadata:
  name: governance-policy-framework
  namespace: open-cluster-management
spec:
  customizedVariables:
    - name: complianceHistoryAPIURL
      value: <replace with URL from previous command>
```

- Replace the **value** parameter value with your compliance history external URL.

2.3.5.3.1. Enabling on all managed clusters

Enable the compliance history API on all managed clusters to record compliance events from your managed clusters. Complete the following steps:

1. Configure the **governance-policy-framework ClusterManagementAddOn** object to use the **AddOnDeploymentConfig** with the following command:

```
oc edit ClusterManagementAddOn governance-policy-framework
```

2. Add or update the **spec.supportedConfigs** array. Your resource might have the following configuration:

```
 - group: addon.open-cluster-management.io
   resource: addondeploymentconfigs
   defaultConfig:
     name: governance-policy-framework
     namespace: open-cluster-management
```

2.3.5.3.2. Enabling compliance history on a single managed cluster

Enable the compliance history API on a single managed cluster to record compliance events from the managed cluster. Complete the following steps:

1. Configure the **governance-policy-framework ManagedClusterAddOn** resource in the managed cluster namespace. Run the following command from your Red Hat Advanced Cluster Management hub cluster with the following command:

```
oc -n <manage-cluster-namespace> edit ManagedClusterAddOn governance-policy-framework
```

   - Replace the **<manage-cluster-namespace>** placeholder with the managed cluster name you intend to enable.

2. Add or update the **spec.configs** array to have an entry similar to the following example:

```
- group: addon.open-cluster-management.io
  resource: addondeploymentconfigs
  name: governance-policy-framework
  namespace: open-cluster-management
```

3. To verify the configuration, confirm that the deployment on your managed cluster is using the **--compliance-api-url** container argument. Run the following command:

```
oc -n open-cluster-management-agent-addon get deployment governance-policy-framework -o jsonpath='{.spec.template.spec.containers[1].args}'
```

   The output might resemble the following:

```
["--enable-lease=true","--hub-cluster-configfile=/var/run/klusterlet/kubeconfig","--leader-elect=false","--log-encoder=console","--log-level=0","--v=-1","--evaluation-concurrency=2","--client-max-qps=30","--client-burst=45","--disable-spec-sync=true","--cluster-namespace=local-cluster","--compliance-api-url=https://governance-history-api-open-cluster-management.apps.openshift.redhat.com"]
```

   Any new policy compliance events are recorded in the policy compliance history API.

   a. If policy compliance events are not being recorded for a specific managed cluster, view the **governance-policy-framework** logs on the affected managed cluster:

```
oc -n open-cluster-management-agent-addon logs deployment/governance-policy-framework -f
```

   b. Log messages similar to the following message are displayed. If the **message** value is empty, the policy compliance history API URL is incorrect or there is a network communication issue:

```
024-03-05T19:28:38.063Z        info    policy-status-sync      statussync/policy_status_sync.go:750    Failed to record the compliance event with the compliance API. Will requeue.       {"statusCode": 503, "message": ""}
```

   c. If the policy compliance history API URL is incorrect, edit the URL on the hub cluster with the following command:

```
oc -n open-cluster-management edit AddOnDeploymentConfig governance-policy-framework
```

+ **Note:** If you experience a network communication issue, you must diagnose the problem based on your network infrastructure.

2.3.5.4. Additional resource

- See Policy compliance history API (Technology Preview).
- See Hub cluster policy framework

2.3.6. Integrating Policy Generator

By integrating the Policy Generator, you can use it to automatically build Red Hat Advanced Cluster Management for Kubernetes policies. To integrate the Policy Generator, see Policy Generator.

For an example of what you can do with the Policy Generator, see Generating a policy that installs the Compliance Operator.

2.3.7. Policy Generator

The Policy Generator is a part of the Red Hat Advanced Cluster Management for Kubernetes application lifecycle subscription Red Hat OpenShift GitOps workflow that generates Red Hat Advanced Cluster Management policies using Kustomize. The Policy Generator builds Red Hat Advanced Cluster Management policies from Kubernetes manifest YAML files, which are provided through a **PolicyGenerator** manifest YAML file that is used to configure it. The Policy Generator is implemented as a Kustomize generator plug-in. For more information on Kustomize, read the *Kustomize documentation*.

View the following sections for more information about the Policy Generator:

- Policy Generator capabilities
- Policy Generator configuration structure
- Policy Generator configuration reference table

2.3.7.1. Policy Generator capabilities

The integration of the Policy Generator with the Red Hat Advanced Cluster Management application lifecycle subscription OpenShift GitOps workflow simplifies the distribution of Kubernetes resource objects to managed OpenShift Container Platform clusters, and Kubernetes clusters through Red Hat Advanced Cluster Management policies.

Use the Policy Generator to complete the following actions:

- Convert any Kubernetes manifest files to Red Hat Advanced Cluster Management configuration policies, including manifests that are created from a Kustomize directory.
- Patch the input Kubernetes manifests before they are inserted into a generated Red Hat Advanced Cluster Management policy.
- Generate additional configuration policies so you can report on Gatekeeper policy violations through Red Hat Advanced Cluster Management for Kubernetes.
- Generate policy sets on the hub cluster.

2.3.7.2. Policy Generator configuration structure

The Policy Generator is a Kustomize generator plug-in that is configured with a manifest of the **PolicyGenerator** kind and **policy.open-cluster-management.io/v1** API version. Continue reading to learn about the configuration structure:

- To use the plug-in, add a **generators** section in a **kustomization.yaml** file. View the following example:

```
generators:
  - policy-generator-config.yaml 1
```

1 The **policy-generator-config.yaml** file that is referenced in the previous example is a YAML file with the instructions of the policies to generate.

- A simple **PolicyGenerator** configuration file might resemble the following example, with **policyDefaults** and **policies**:

```
apiVersion: policy.open-cluster-management.io/v1
kind: PolicyGenerator
metadata:
  name: config-data-policies
policyDefaults:
  namespace: policies
  policySets: []
policies:
  - name: config-data
    manifests:
      - path: configmap.yaml 1
```

1 The **configmap.yaml** file represents a Kubernetes manifest YAML file to be included in the policy. Alternatively, you can set the path to a Kustomize directory, or a directory with multiple Kubernetes manifest YAML files. View the following **ConfigMap** example:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: my-config
  namespace: default
data:
  key1: value1
  key2: value2
```

- You can use the **object-templates-raw** manifest to automatically generate a **ConfigurationPolicy** resource with the content you add. For example, you can create a manifest file with the following syntax:

```
object-templates-raw: |
  {{- range (lookup "v1" "ConfigMap" "my-namespace" "").items }}
  - complianceType: musthave
    objectDefinition:
      kind: ConfigMap
      apiVersion: v1
```

```
    metadata:
      name: {{ .metadata.name }}
      namespace: {{ .metadata.namespace }}
      labels:
        i-am-from: template
  {{- end }}
```

- After you create a manifest file, you can create a **PolicyGenerator** configuration file. See the following YAML example and for **path**, enter the path to your **manifest.yaml** file:

```
apiVersion: policy.open-cluster-management.io/v1
kind: PolicyGenerator
metadata:
  name: config-data-policies
policyDefaults:
  namespace: policies
  policySets: []
policies:
  - name: config-data
    manifests:
      - path: manifest.yaml
```

- The generated **Policy** resource, along with the generated **Placement** resource, and **PlacementBinding** resource, might resemble the following examples. **Note:** Specifications for resources are described in the *Policy Generator configuration reference table*.

```
apiVersion: cluster.open-cluster-management.io/v1beta1
kind: Placement
metadata:
  name: placement-config-data
  namespace: policies
spec:
  predicates:
  - requiredClusterSelector:
      labelSelector:
        matchExpressions: []
    tolerations:
      - key: cluster.open-cluster-management.io/unavailable
        operator: Exists
      - key: cluster.open-cluster-management.io/unreachable
        operator: Exists
---
apiVersion: policy.open-cluster-management.io/v1
kind: PlacementBinding
metadata:
  name: binding-config-data
  namespace: policies
placementRef:
  apiGroup: cluster.open-cluster-management.io
  kind: Placement
  name: placement-config-data
subjects:
- apiGroup: policy.open-cluster-management.io
  kind: Policy
  name: config-data
```

```
---
apiVersion: policy.open-cluster-management.io/v1
kind: Policy
metadata:
  annotations:
    policy.open-cluster-management.io/categories: CM Configuration Management
    policy.open-cluster-management.io/controls: CM-2 Baseline Configuration
    policy.open-cluster-management.io/standards: NIST SP 800-53
    policy.open-cluster-management.io/description:
  name: config-data
  namespace: policies
spec:
  disabled: false
  policy-templates:
  - objectDefinition:
      apiVersion: policy.open-cluster-management.io/v1
      kind: ConfigurationPolicy
      metadata:
        name: config-data
      spec:
        object-templates:
        - complianceType: musthave
          objectDefinition:
            apiVersion: v1
            data:
              key1: value1
              key2: value2
            kind: ConfigMap
            metadata:
              name: my-config
              namespace: default
        remediationAction: inform
        severity: low
```

2.3.7.3. Policy Generator configuration reference table

All the fields in the **policyDefaults** section, except for **namespace**, can be overridden for each policy, and all the fields in the **policySetDefaults** section can be overridden for each policy set.

Table 2.2. Parameter table

Field	Optional or required	Description
apiVersion	Required	Set the value to **policy.open-cluster-management.io/v1**.
kind	Required	Set the value to **PolicyGenerator** to indicate the type of policy.
metadata.name	Required	The name for identifying the policy resource.

Field	Optional or required	Description
placementBindingDefaults.name	Optional	If multiple policies use the same placement, this name is used to generate a unique name for the resulting **PlacementBinding**, binding the placement with the array of policies that reference it.
policyDefaults	Required	Any default value listed here is overridden by an entry in the policies array except for **namespace**.
policyDefaults.namespace	Required	The namespace of all the policies.
policyDefaults.complianceType	Optional	Determines the policy controller behavior when comparing the manifest to objects on the cluster. The values that you can use are **musthave**, **mustonlyhave**, or **mustnothave**. The default value is **musthave**.
policyDefaults.metadataComplianceType	Optional	Overrides **complianceType** when comparing the manifest metadata section to objects on the cluster. The values that you can use are **musthave**, and **mustonlyhave**. The default value is empty (**{}**) to avoid overriding the **complianceType** for metadata.
policyDefaults.categories	Optional	Array of categories to be used in the **policy.open-cluster-management.io/categories** annotation. The default value is **CM Configuration Management**.
policyDefaults.controls	Optional	Array of controls to be used in the **policy.open-cluster-management.io/controls** annotation. The default value is **CM-2 Baseline Configuration**.

Field	Optional or required	Description
policyDefaults.standards	Optional	An array of standards to be used in the **policy.open-cluster-management.io/standards** annotation. The default value is **NIST SP 800-53**.
policyDefaults.policyAnnotations	Optional	Annotations that the policy includes in the **metadata.annotations** section. It is applied for all policies unless specified in the policy. The default value is empty (**{}**).
policyDefaults.configurationPolicyAnnotations	Optional	Key-value pairs of annotations to set on generated configuration policies. For example, you can disable policy templates by defining the following parameter: **{"policy.open-cluster-management.io/disable-templates": "true"}**. The default value is empty (**{}**).
policyDefaults.copyPolicyMetadata	Optional	Copies the labels and annotations for all policies and adds them to a replica policy. Set to **true** by default. If set to **false**, only the policy framework specific policy labels and annotations are copied to the replicated policy.
policyDefaults.customMessage	Optional	Configures the compliance messages emitted by the configuration policy to use one of the specified Go templates based on the current compliance.
policyDefaults.severity	Optional	The severity of the policy violation. The default value is **low**.
policyDefaults.disabled	Optional	Whether the policy is disabled, meaning it is not propagated and no status as a result. The default value is **false** to enable the policy.
policyDefaults.remediationAction	Optional	The remediation mechanism of your policy. The parameter values are **enforce** and **inform**. The default value is **inform**.

Field	Optional or required	Description
policyDefaults.namespaceSelector	Required for namespaced objects that do not have a namespace specified	Determines namespaces in the managed cluster that the object is applied to. The **include** and **exclude** parameters accept file path expressions to include and exclude namespaces by name. The **matchExpressions** and **matchLabels** parameters specify namespaces to include by label. Read the *Kubernetes labels and selectors* documentation. The resulting list is compiled by using the intersection of results from all parameters.
policyDefaults.evaluationInterval	Optional	Specifies the frequency for a policy to be evaluated when it is in a particular compliance state. Use the parameters **compliant** and **noncompliant**. The default value for the **compliant** and **noncompliant** parameters is **watch** to leverage Kubernetes API watches instead of polling the Kubernetes API server. When managed clusters have low resources, the evaluation interval can be set to long polling intervals to reduce CPU and memory usage on the Kubernetes API and policy controller. These are in the format of durations. For example, **1h25m3s** represents 1 hour, 25 minutes, and 3 seconds. These can also be set to **never** to avoid evaluating the policy after it is in a particular compliance state.
policyDefaults.evaluationInterval.compliant	Optional	Specifies the evaluation frequency for a compliant policy. If you want to enable the previous polling behavior, set this parameter to **10s**.
policyDefaults.evaluationInterval.noncompliant	Optional	Specifies the evaluation frequency for a non-compliant policy. If you want to enable the previous polling behavior, set this parameter to **10s**.

Field	Optional or required	Description
policyDefaults.pruneObjectBehavior	Optional	Determines whether objects created or monitored by the policy should be deleted when the policy is deleted. Pruning only takes place if the remediation action of the policy has been set to **enforce**. Example values are **DeleteIfCreated**, **DeleteAll**, or **None**. The default value is **None**.
policyDefaults.recreateOption	Optional	Describes whether to delete and recreate an object when an update is required. The **IfRequired** value recreates the object when you update an immutable field. Without dry run update support, the **IfRequired** has no effect on clusters. The **Always** value always recreate the object if a mismatch is detected. When the **remediationAction** parameter is set to **inform** the **RecreateOption** value has no effect. The default value is **None**.
policyDefaults.recordDiff	Optional	Specifies if and where to log the difference between the object on the cluster and the **objectDefinition** in the policy. Set to **Log** to log the difference in the controller logs or **None** to not log the difference. By default, this parameter is empty to not log the difference.
policyDefaults.dependencies	Optional	A list of objects that must be in specific compliance states before this policy is applied. Cannot be specified when **policyDefaults.orderPolicies** is set to **true**.

Field	Optional or required	Description
policyDefaults.dependencies[].name	Required	The name of the object being depended on.
policyDefaults.dependencies[].namespace	Optional	The namespace of the object being depended on. The default is the namespace of policies set for the Policy Generator.
policyDefaults.dependencies[].compliance	Optional	The compliance state the object needs to be in. The default value is **Compliant**.
policyDefaults.dependencies[].kind	Optional	The kind of the object. By default, the kind is set to **Policy**, but can also be other kinds that have compliance state, such as **ConfigurationPolicy**.
policyDefaults.dependencies[].apiVersion	Optional	The API version of the object. The default value is **policy.open-cluster-management.io/v1**.
policyDefaults.description	Optional	The description of the policy you want to create.
policyDefaults.extraDependencies	Optional	A list of objects that must be in specific compliance states before this policy is applied. The dependencies that you define are added to each policy template (for example, **ConfigurationPolicy**) separately from the **dependencies** list. Cannot be specified when **policyDefaults.orderManifests** is set to **true**.
policyDefaults.extraDependencies[].name	Required	The name of the object being depended on.
policyDefaults.extraDependencies[].namespace	Optional	The namespace of the object being depended on. By default, the value is set to the namespace of policies set for the Policy Generator.

Field	Optional or required	Description
policyDefaults.extraDependencies[].compliance	Optional	The compliance state the object needs to be in. The default value is **Compliant**.
policyDefaults.extraDependencies[].kind	Optional	The kind of the object. The default value is to **Policy**, but can also be other kinds that have a compliance state, such as **ConfigurationPolicy**.
policyDefaults.extraDependencies[].apiVersion	Optional	The API version of the object. The default value is **policy.open-cluster-management.io/v1**.
policyDefaults.ignorePending	Optional	Bypass compliance status checks when the Policy Generator is waiting for its dependencies to reach their desired states. The default value is **false**.
policyDefaults.orderPolicies	Optional	Automatically generate **dependencies** on the policies so they are applied in the order you defined in the policies list. By default, the value is set to **false**. Cannot be specified at the same time as **policyDefaults.dependencies**.
policyDefaults.orderManifests	Optional	Automatically generate **extraDependencies** on policy templates so that they are applied in the order you defined in the manifests list for that policy. Cannot be specified when **policyDefaults.consolidateManifests** is set to **true**. Cannot be specified at the same time as **policyDefaults.extraDependencies**.
policyDefaults.consolidateManifests	Optional	This determines if a single configuration policy is generated for all the manifests being wrapped in the policy. If set to **false**, a configuration policy per manifest is generated. The default value is **true**.

Field	Optional or required	Description
policyDefaults.informGatekeeperPolicies (Deprecated)	Optional	Set **informGatekeeperPolicies** to false to use Gatekeeper manifests directly without defining it in a configuration policy. When the policy references a violated Gatekeeper policy manifest, an additional configuration policy is generated in order to receive policy violations in Red Hat Advanced Cluster Management. The default value is **true**.
policyDefaults.informKyvernoPolicies	Optional	When the policy references a Kyverno policy manifest, this determines if an additional configuration policy is generated to receive policy violations in Red Hat Advanced Cluster Management, when the Kyverno policy is violated. The default value is **true**.
policyDefaults.policyLabels	Optional	Labels that the policy includes in its **metadata.labels** section. The **policyLabels** parameter is applied for all policies unless specified in the policy.
policyDefaults.gatekeeperEnforcementAction	Optional	Overrides the **spec.enforcementAction** field of a Gatekeeper constraint. This only applies to Gatekeeper constraints and is ignored by other manifests. If not set, the **spec.enforcementAction** field does not change.

Field	Optional or required	Description
policyDefaults.policySets	Optional	Array of policy sets that the policy joins. Policy set details can be defined in the **policySets** section. When a policy is part of a policy set, a placement binding is not generated for the policy since one is generated for the set. Set **policies[].generatePlacementWhenInSet** or **policyDefaults.generatePlacementWhenInSet** to override **policyDefaults.policySets**.
policyDefaults.generatePolicyPlacement	Optional	Generate placement manifests for policies. Set to **true** by default. When set to **false**, the placement manifest generation is skipped, even if a placement is specified.
policyDefaults.generatePlacementWhenInSet	Optional	When a policy is part of a policy set, by default, the generator does not generate the placement for this policy since a placement is generated for the policy set. Set **generatePlacementWhenInSet** to **true** to deploy the policy with both policy placement and policy set placement. The default value is **false**.
policyDefaults.placement	Optional	The placement configuration for the policies. This defaults to a placement configuration that matches all clusters.
policyDefaults.placement.name	Optional	Specifying a name to consolidate placements that contain the same cluster label selectors.

Field	Optional or required	Description
policyDefaults.placement.labelSelector	Optional	Specify a placement by defining a cluster label selector using either **key:value**, or providing a **matchExpressions**, **matchLabels**, or both, with appropriate values. See ***placementPath*** to specify an existing file.
policyDefaults.placement.placementName	Optional	Define this parameter to use a placement that already exists on the cluster. A **Placement** is not created, but a **PlacementBinding** binds the policy to this **Placement**.
policyDefaults.placement.placementPath	Optional	To reuse an existing placement, specify the path relative to the location of the **kustomization.yaml** file. If provided, this placement is used by all policies by default. See ***labelSelector*** to generate a new **Placement**.
policyDefaults.placement.clusterSelector (Deprecated)	Optional	**PlacementRule** is deprecated. Use **labelSelector** instead to generate a placement. Specify a placement rule by defining a cluster selector using either **key:value** or by providing **matchExpressions**, **matchLabels**, or both, with appropriate values. See **placementRulePath** to specify an existing file.

Field	Optional or required	Description
policyDefaults.placement.placementRuleName (Deprecated)	Optional	**PlacementRule** is deprecated. Alternatively, use **placementName** to specify a placement. To use an existing placement rule on the cluster, specify the name for this parameter. A **PlacementRule** is not created, but a **PlacementBinding** binds the policy to the existing **PlacementRule**.
policyDefaults.placement.placementRulePath (Deprecated)	Optional	**PlacementRule** is deprecated. Alternatively, use **placementPath** to specify a placement. To reuse an existing placement rule, specify the path relative to the location of the **kustomization.yaml** file. If provided, this placement rule is used by all policies by default. See ***clusterSelector*** to generate a new **PlacementRule**.
policySetDefaults	Optional	Default values for policy sets. Any default value listed for this parameter is overridden by an entry in the **policySets** array.
policySetDefaults.placement	Optional	The placement configuration for the policies. This defaults to a placement configuration that matches all clusters. See ***policyDefaults.placement*** for description of this field.
policySetDefaults.generatePolicySetPlacement	Optional	Generate placement manifests for policy sets. Set to **true** by default. When set to **false** the placement manifest generation is skipped, even if a placement is specified.
policies	Required	The list of policies to create along with overrides to either the default values, or the values that are set in **policyDefaults**. See ***policyDefaults*** for additional fields and descriptions.

Field	Optional or required	Description
policies.description	Optional	The description of the policy you want to create.
policies[].name	Required	The name of the policy to create.
policies[].manifests	Required	The list of Kubernetes object manifests to include in the policy, along with overrides to either the default values, the values set in this **policies** item, or the values set in **policyDefaults**. See **policyDefaults** for additional fields and descriptions. When **consolidateManifests** is set to **true**, only **complianceType**, **metadataComplianceType**, and **recordDiff** can be overridden at the **policies[].manifests** level.
policies[].manifests[].path	Required	Path to a single file, a flat directory of files, or a Kustomize directory relative to the **kustomization.yaml** file. If the directory is a Kustomize directory, the generator runs Kustomize against the directory before generating the policies. If there is a requirement to process Helm charts for the Kustomize directory, set **POLICY_GEN_ENABLE_HELM** to **true** in the environment where the policy generator is running to enable Helm for the Policy Generator. The following manifests are supported: ● Non-root policy type manifests such as **CertificatePolicy**, **ConfigurationPolicy**, **OperatorPolicy** that have a **Policy** suffix. The previous manifests are not modified except for patches and are directly added as a **policy-templates** entry of the **Policy** resource. Gatekeeper constraints

Field	Optional or required	Description
		are also directly included **informGatekeeperPolicies** is set to **false**. • Manifests containing only an **object-templates-raw** key. The corresponding value is used directly in a generated **ConfigurationPolicy** resource without modification, which is added as a **policy-templates** entry of the **Policy** entry. • For everything else, **ConfigurationPolicy** objects are generated to wrap the previously mentioned manifests. The resulting **ConfigurationPolicy** manifest is added as a **policy-templates** entry of the **Policy** resource.
policies[].manifests[].name	Optional	Serves as the **ConfigurationPolicy** resource name when **ConsolidateManifests** is set to **false**. If multiple manifests are present in the path, an index number is appended. If multiple manifests are present and their names are provided, with **consolidateManifests** set to **true**, the name of the first manifest is used for all manifest paths.

Field	Optional or required	Description
policies[].manifests[].patches	Optional	A list of Kustomize patches to apply to the manifest at the path. If there are multiple manifests, the patch requires the **apiVersion**, **kind**, **metadata.name**, and **metadata.namespace** (if applicable) fields to be set so Kustomize can identify the manifest that the patch applies to. If there is a single manifest, the **metadata.name** and **metadata.namespace** fields can be patched.
policies.policyLabels	Optional	Labels that the policy includes in its **metadata.labels** section. The **policyLabels** parameter is applied for all policies unless specified in the policy.
policySets	Optional	The list of policy sets to create, along with overrides to either the default values or the values that are set in **policySetDefaults**. To include a policy in a policy set, use **policyDefaults.policySets**, **policies[].policySets**, or **policySets.policies**. See ***policySetDefaults*** for additional fields and descriptions.
policySets[].name	Required	The name of the policy set to create.
policySets[].description	Optional	The description of the policy set to create.
policySets[].policies	Optional	The list of policies to be included in the policy set. If **policyDefaults.policySets** or **policies[].policySets** is also specified, the lists are merged.

2.3.7.4. Additional resources

- Read Generating a policy to install GitOps Operator .
- Read to Policy set controller for more details.

- Read Applying Kustomize for more information.
- Read the Governance documentation for more topics.
- See an example of a **kustomization.yaml** file.
- Refer to the Kubernetes labels and selectors documentation.
- Refer to Gatekeeper for more details.
- Refer to the Kustomize documentation.

2.3.8. Generating a policy that installs the Compliance Operator

Generate a policy that installs the Compliance Operator onto your clusters. For an operator that uses the *namespaced* installation mode, such as the Compliance Operator, an **OperatorGroup** manifest is also required.

Complete the following steps:

1. Create a YAML file with a **Namespace**, a **Subscription**, and an **OperatorGroup** manifest called **compliance-operator.yaml**. The following example installs these manifests in the **compliance-operator** namespace:

```
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-compliance
---
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: compliance-operator
  namespace: openshift-compliance
spec:
  targetNamespaces:
    - openshift-compliance
---
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: compliance-operator
  namespace: openshift-compliance
spec:
  channel: release-0.1
  name: compliance-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
```

2. Create a **PolicyGenerator** configuration file. View the following **PolicyGenerator** policy example that installs the Compliance Operator on all OpenShift Container Platform managed clusters:

```
apiVersion: policy.open-cluster-management.io/v1
kind: PolicyGenerator
```

```
metadata:
  name: install-compliance-operator
policyDefaults:
  namespace: policies
  placement:
    labelSelector:
      matchExpressions:
        - key: vendor
          operator: In
          values:
            - "OpenShift"
policies:
  - name: install-compliance-operator
    manifests:
      - path: compliance-operator.yaml
```

3. Add the policy generator to your **kustomization.yaml** file. The **generators** section might resemble the following configuration:

```
generators:
  - policy-generator-config.yaml
```

As a result, the generated policy resembles the following file:

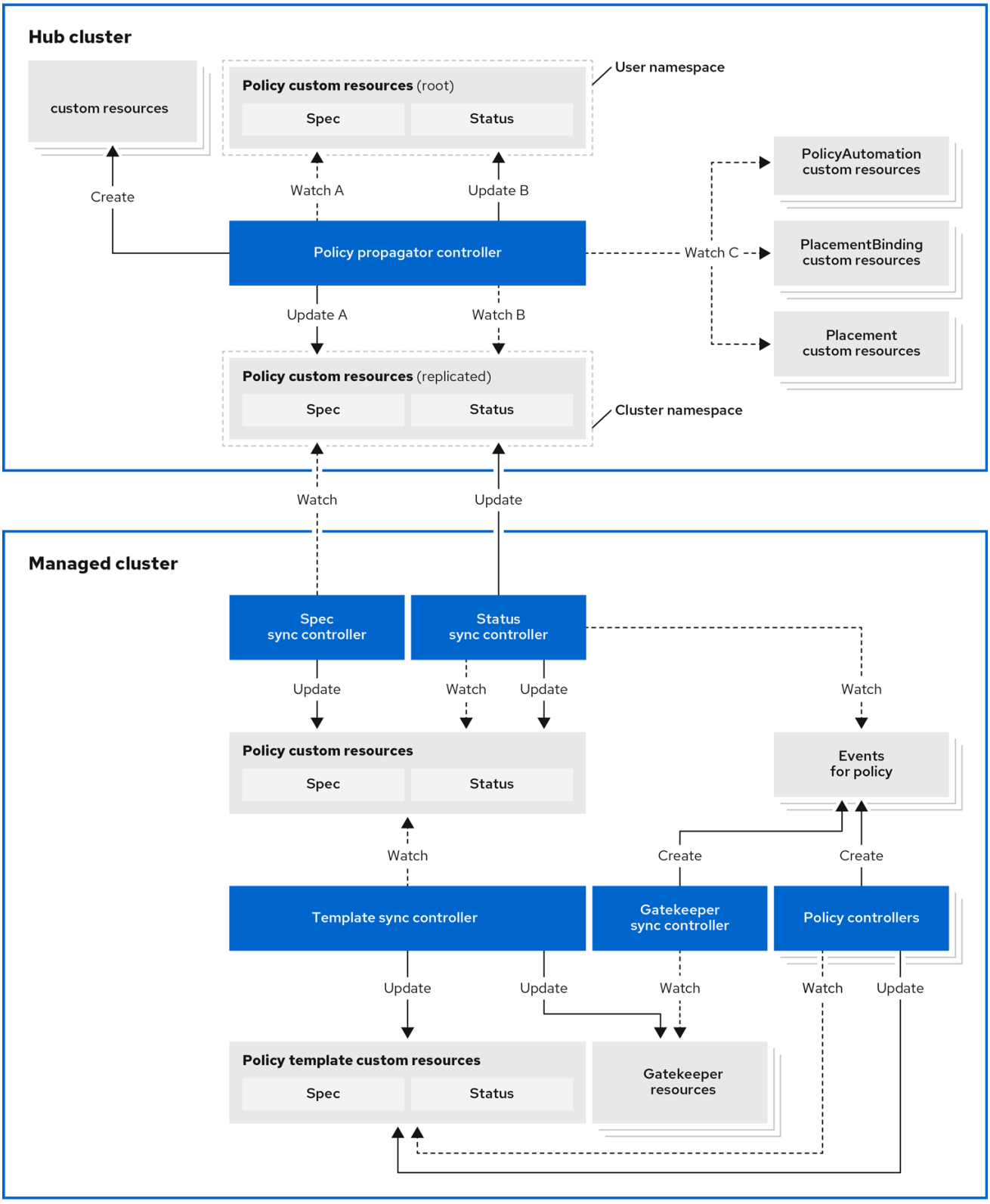
```
apiVersion: cluster.open-cluster-management.io/v1beta1
kind: Placement
metadata:
  name: placement-install-compliance-operator
  namespace: policies
spec:
  predicates:
  - requiredClusterSelector:
      labelSelector:
        matchExpressions:
        - key: vendor
          operator: In
          values:
          - OpenShift
      tolerations:
        - key: cluster.open-cluster-management.io/unavailable
          operator: Exists
        - key: cluster.open-cluster-management.io/unreachable
          operator: Exists
---
apiVersion: policy.open-cluster-management.io/v1
kind: PlacementBinding
metadata:
  name: binding-install-compliance-operator
  namespace: policies
placementRef:
  apiGroup: cluster.open-cluster-management.io
  kind: Placement
  name: placement-install-compliance-operator
subjects:
  - apiGroup: policy.open-cluster-management.io
```

```
    kind: Policy
    name: install-compliance-operator
---
apiVersion: policy.open-cluster-management.io/v1
kind: Policy
metadata:
  annotations:
    policy.open-cluster-management.io/categories: CM Configuration Management
    policy.open-cluster-management.io/controls: CM-2 Baseline Configuration
    policy.open-cluster-management.io/standards: NIST SP 800-53
    policy.open-cluster-management.io/description:
  name: install-compliance-operator
  namespace: policies
spec:
  disabled: false
  policy-templates:
    - objectDefinition:
        apiVersion: policy.open-cluster-management.io/v1
        kind: ConfigurationPolicy
        metadata:
          name: install-compliance-operator
        spec:
          object-templates:
            - complianceType: musthave
              objectDefinition:
                apiVersion: v1
                kind: Namespace
                metadata:
                  name: openshift-compliance
            - complianceType: musthave
              objectDefinition:
                apiVersion: operators.coreos.com/v1alpha1
                kind: Subscription
                metadata:
                  name: compliance-operator
                  namespace: openshift-compliance
                spec:
                  channel: release-0.1
                  name: compliance-operator
                  source: redhat-operators
                  sourceNamespace: openshift-marketplace
            - complianceType: musthave
              objectDefinition:
                apiVersion: operators.coreos.com/v1
                kind: OperatorGroup
                metadata:
                  name: compliance-operator
                  namespace: openshift-compliance
                spec:
                  targetNamespaces:
                    - compliance-operator
          remediationAction: enforce
          severity: low
```

As a result, the generated policy is displayed.

2.3.9. Governance policy framework architecture

Enhance the security for your cluster with the Red Hat Advanced Cluster Management for Kubernetes governance lifecycle. The product governance lifecycle is based on using supported policies, processes, and procedures to manage security and compliance from a central interface page. View the following diagram of the governance architecture:



View the following component descriptions for the governance architecture diagram:

- **Governance policy framework:** The previous image represents the framework that runs as the **governance-policy-framework** pod on managed clusters and contains the following controllers:
    - **Spec sync controller:** Synchronizes the replicated policy in the managed cluster namespace on the hub cluster to the managed cluster namespace on the managed cluster.
    - **Status sync controller:** Records compliance events from policy controllers in the replicated policies on the hub and managed cluster. The status only contains updates that are relevant to the current policy and does not consider past statuses if the policy is deleted and recreated.
    - **Template sync controller:** Creates, updates, and deletes objects in the managed cluster namespace on the managed cluster based on the definitions from the replicated policy **spec.policy-templates** entries.
    - **Gatekeeper sync controller:** Records Gatekeeper constraint audit results as compliance events in corresponding Red Hat Advanced Cluster Management policies.
- **Policy propagator controller:** Runs on the Red Hat Advanced Cluster Management hub cluster and generates the replicated policies in the managed cluster namespaces on the hub based on the placements bound to the root policy. It also aggregates compliance status from replicated policies to the root policy status and initiates automations based on policy automations bound to the root policy.
- **Governance policy add-on controller:** Runs on the Red Hat Advanced Cluster Management hub cluster and manages the installation of policy controllers on managed clusters.

2.3.9.1. Governance architecture components

The governance architecture also include following components:

- **Governance dashboard:** Provides a summary of your cloud governance and risk details, which include policy and cluster violations. Refer to the *Manage Governance dashboard* section to learn about the structure of an Red Hat Advanced Cluster Management for Kubernetes policy framework, and how to use the Red Hat Advanced Cluster Management for Kubernetes *Governance* dashboard.
  Notes:
    - When a policy is propagated to a managed cluster, it is first replicated to the cluster namespace on the hub cluster, and is named and labeled using **namespaceName.policyName**. When you create a policy, make sure that the length of the **namespaceName.policyName** does not exceed 63 characters due to the Kubernetes length limit for label values.
    - When you search for a policy in the hub cluster, you might also receive the name of the replicated policy in the managed cluster namespace. For example, if you search for **policy-dhaz-cert** in the **default** namespace, the following policy name from the hub cluster might also appear in the managed cluster namespace: **default.policy-dhaz-cert**.
- **Policy-based governance framework:** Supports policy creation and deployment to various managed clusters based on attributes associated with clusters, such as a geographical region. There are examples of the predefined policies and instructions on deploying policies to your cluster in the *open source community*. Additionally, when policies are violated, automations can be configured to run and take any action that the user chooses.

- **Open source community:** Supports community contributions with a foundation of the Red Hat Advanced Cluster Management policy framework. Policy controllers and third-party policies are also a part of the **open-cluster-management/policy-collection** repository. You can contribute and deploy policies using GitOps.

2.3.9.2. Additional resources

- See Policy controllers introduction.
- See Deploying policies by using GitOps .

2.3.10. Governance dashboard

Manage your security policies and policy violations by using the *Governance* dashboard to create, view, and edit your resources. You can create YAML files for your policies from the command line and console. Continue reading for details about the *Governance* dashboard from the console.

2.3.10.1. Governance page

The following tabs are displayed on the *Governance* page *Overview*, *Policy sets*, and *Policies*. Read the following descriptions to know which information is displayed:

- **Overview**
  The following summary cards are displayed from the *Overview* tab: *Policy set violations*, *Policy violations*, *Clusters*, *Categories*, *Controls*, and *Standards*.
- **Policy sets**
  Create and manage hub cluster policy sets.
- **Policies**
  - Create and manage security policies. The table of policies list the following details of a policy: *Name*, *Namespace*, and *Cluster violations* are displayed.
  - You can edit, enable or disable, set remediation to inform or enforce, or remove a policy by selecting the **Actions** icon. You can view the categories and standards of a specific policy by selecting the drop-down arrow to expand the row.
  - Reorder your table columns in the *Manage column* dialog box. Select the *Manage column* icon for the dialog box to be displayed. To reorder your columns, select the *Reorder* icon and move the column name. For columns that you want to appear in the table, click the checkbox for specific column names and select the **Save** button.
  - Complete bulk actions by selecting multiple policies and clicking the **Actions** button. You can also customize your policy table by clicking the **Filter** button.
    When you select a policy in the table list, the following tabs of information are displayed from the console:
    - *Details*: Select the *Details* tab to view policy details and placement details. In the *Placement* table, the *Compliance* column provides links to view the compliance of the clusters that are displayed.
    - *Results*: Select the *Results* tab to view a table list of all clusters that are associated to the policy.

- From the *Message* column, click the **View details** link to view the template details, template YAML, and related resources. You can also view related resources. Click the **View history** link to view the violation message and a time of the last report.

2.3.10.2. Governance automation configuration

If there is a configured automation for a specific policy, you can select the automation to view more details. View the following descriptions of the schedule frequency options for your automation:

- *Manual run*: Manually set this automation to run once. After the automation runs, it is set to **disabled**. **Note:** You can only select *Manual run* mode when the schedule frequency is disabled.

- *Run once mode* : When a policy is violated, the automation runs one time. After the automation runs, it is set to **disabled**. After the automation is set to **disabled**, you must continue to run the automation manually. When you run *once mode*, the extra variable of **target_clusters** is automatically supplied with the list of clusters that violated the policy. The Ansible Automation Platform Job template must have **PROMPT ON LAUNCH** enabled for the **EXTRA VARIABLES** section (also known as **extra_vars**).

- *Run everyEvent mode* : When a policy is violated, the automation runs every time for each unique policy violation per managed cluster. Use the **DelayAfterRunSeconds** parameter to set the minimum seconds before an automation can be restarted on the same cluster. If the policy is violated multiple times during the delay period and kept in the violated state, the automation runs one time after the delay period. The default is 0 seconds and is only applicable for the **everyEvent** mode. When you run **everyEvent** mode, the extra variable of **target_clusters** and Ansible Automation Platform Job template is the same as *once mode*.

- *Disable automation*: When the scheduled automation is set to **disabled**, the automation does not run until the setting is updated.

The following variables are automatically provided in the **extra_vars** of the Ansible Automation Platform Job:

- **policy_name**: The name of the non-compliant root policy that initiates the Ansible Automation Platform job on the hub cluster.

- **policy_namespace**: The namespace of the root policy.

- **hub_cluster**: The name of the hub cluster, which is determined by the value in the **clusters DNS** object.

- **policy_sets**: This parameter contains all associated policy set names of the root policy. If the policy is not within a policy set, the **policy_set** parameter is empty.

- **policy_violations**: This parameter contains a list of non-compliant cluster names, and the value is the policy **status** field for each non-compliant cluster.

2.3.10.3. Additional resources

Review the following topics to learn more about creating and updating your security policies:

- Managing security policies

- Creating configuration policies

- Managing the Gatekeeper operator installation policies

- Configuring Ansible Automation Platform for governance
- Governance

2.3.11. Creating configuration policies

You can create a YAML file for your configuration policy from the command line interface (CLI) or from the console. As you create a configuration policy from the console, a YAML file is displayed in the YAML editor.

2.3.11.1. Prerequisites

- **Required access**: Administrator or cluster administrator
- If you have an existing Kubernetes manifest, consider using the Policy Generator to automatically include the manifests in a policy. See the Policy Generator documentation.

2.3.11.2. Creating a configuration policy from the CLI

Complete the following steps to create a configuration policy from the (CLI):

1. Create a YAML file for your configuration policy. Run the following command:

```
oc create -f configpolicy-1.yaml
```

   Your configuration policy might resemble the following policy:

```
apiVersion: policy.open-cluster-management.io/v1
kind: Policy
metadata:
  name: policy-1
  namespace: my-policies
policy-templates:
- apiVersion: policy.open-cluster-management.io/v1
  kind: ConfigurationPolicy
  metadata:
    name: mustonlyhave-configuration
  spec:
    namespaceSelector:
      include: ["default"]
      exclude: ["kube-system"]
    remediationAction: inform
    disabled: false
    complianceType: mustonlyhave
    object-templates:
```

2. Apply the policy by running the following command:

```
oc apply -f <policy-file-name>  --namespace=<namespace>
```

3. Verify and list the policies by running the following command:

```
oc get policies.policy.open-cluster-management.io --namespace=<namespace>
```

Your configuration policy is created.

2.3.11.2.1. Viewing your configuration policy from the CLI

Complete the following steps to view your configuration policy from the CLI:

1. View details for a specific configuration policy by running the following command:

```
oc get policies.policy.open-cluster-management.io <policy-name> -n <namespace> -o yaml
```

2. View a description of your configuration policy by running the following command:

```
oc describe policies.policy.open-cluster-management.io <name> -n <namespace>
```

2.3.11.2.2. Viewing your configuration policy from the console

View any configuration policy and its status from the console.

After you log in to your cluster from the console, select **Governance** to view a table list of your policies. **Note:** You can filter the table list of your policies by selecting the *All policies* tab or *Cluster violations* tab.

Select one of your policies to view more details. The *Details*, *Clusters*, and *Templates* tabs are displayed.

2.3.11.3. Disabling configuration policies

To disable your configuration policy, select **Disable policy** from the *Actions* menu of the policy. The policy is disabled, but not deleted.

2.3.11.4. Deleting a configuration policy

Delete a configuration policy from the CLI or the console. Complete the following steps:

1. To delete the policy from your target cluster or clusters, run the following command:

```
oc delete policies.policy.open-cluster-management.io <policy-name> -n <namespace>
```

2. Verify that your policy is removed by running the following command:

```
oc get policies.policy.open-cluster-management.io <policy-name> -n <namespace>
```

3. To delete a configuration policy from the console, click the **Actions** icon for the policy you want to delete in the policy violation table and then click **Delete**.

Your policy is deleted.

2.3.11.5. Additional resources

- See configuration policy samples that are supported by Red Hat Advanced Cluster Management from the CM-Configuration-Management folder.
- Alternatively, you can refer to the Table of sample configuration policies to view other configuration policies that are monitored by the controller.

2.3.12. Configuring Ansible Automation Platform for governance

Red Hat Advanced Cluster Management for Kubernetes governance can be integrated with Red Hat Ansible Automation Platform to create policy violation automations. You can configure the automation from the Red Hat Advanced Cluster Management console.

- Prerequisites
- Creating a policy violation automation from the console
- Creating a policy violation automation from the CLI

2.3.12.1. Prerequisites

- A supported OpenShift Container Platform version
- You must have Ansible Automation Platform version 3.7.3 or a later version installed. It is best practice to install the latest supported version of Ansible Automation Platform. See Red Hat Ansible Automation Platform documentation for more details.
- Install the Ansible Automation Platform Resource Operator from the Operator Lifecycle Manager. In the *Update Channel* section, select **stable-2.x-cluster-scoped**. Select the **All namespaces on the cluster (default)** installation mode.
  Note: Ensure that the Ansible Automation Platform job template is idempotent when you run it. If you do not have Ansible Automation Platform Resource Operator, you can find it from the Red Hat OpenShift Container Platform *OperatorHub* page.

For more information about installing and configuring Red Hat Ansible Automation Platform, see Setting up Ansible tasks.

2.3.12.2. Creating a policy violation automation from the console

After you log in to your Red Hat Advanced Cluster Management hub cluster, select **Governance** from the navigation menu, and then click on the *Policies* tab to view the policy tables.

Configure an automation for a specific policy by clicking **Configure** in the *Automation* column. You can create automation when the policy automation panel appears. From the *Ansible credential* section, click the drop-down menu to select an Ansible credential. If you need to add a credential, see Managing credentials overview.

Note: This credential is copied to the same namespace as the policy. The credential is used by the **AnsibleJob** resource that is created to initiate the automation. Changes to the Ansible credential in the *Credentials* section of the console is automatically updated.

After a credential is selected, click the Ansible job drop-down list to select a job template. In the *Extra variables* section, add the parameter values from the **extra_vars** section of the **PolicyAutomation**. Select the frequency of the automation. You can select *Run once mode* , *Run everyEvent mode* , or *Disable automation*.

Save your policy violation automation by selecting **Submit**. When you select the *View Job* link from the Ansible job details side panel, the link directs you to the job template on the *Search* page. After you successfully create the automation, it is displayed in the *Automation* column.

Note: When you delete a policy that has an associated policy automation, the policy automation is automatically deleted as part of clean up.

Your policy violation automation is created from the console.

2.3.12.3. Creating a policy violation automation from the CLI

Complete the following steps to configure a policy violation automation from the CLI:

1. From your terminal, log in to your Red Hat Advanced Cluster Management hub cluster using the **oc login** command.

2. Find or create a policy that you want to add an automation to. Note the policy name and namespace.

3. Create a **PolicyAutomation** resource using the following sample as a guide:

```
apiVersion: policy.open-cluster-management.io/v1beta1
kind: PolicyAutomation
metadata:
  name: policyname-policy-automation
spec:
  automationDef:
    extra_vars:
      your_var: your_value
    name: Policy Compliance Template
    secret: ansible-tower
    type: AnsibleJob
  mode: disabled
  policyRef: policyname
```

4. The Automation template name in the previous sample is **Policy Compliance Template**. Change that value to match your job template name.

5. In the **extra_vars** section, add any parameters you need to pass to the Automation template.

6. Set the mode to either **once**, **everyEvent**, or **disabled**.

7. Set the **policyRef** to the name of your policy.

8. Create a secret in the same namespace as this **PolicyAutomation** resource that contains the Ansible Automation Platform credential. In the previous example, the secret name is **ansible-tower**. Use the sample from application lifecycle to see how to create the secret.

9. Create the **PolicyAutomation** resource.
   Notes:

   - An immediate run of the policy automation can be initiated by adding the following annotation to the **PolicyAutomation** resource:

```
metadata:
  annotations:
    policy.open-cluster-management.io/rerun: "true"
```

   - When the policy is in **once** mode, the automation runs when the policy is non-compliant. The **extra_vars** variable, named **target_clusters** is added and the value is an array of each managed cluster name where the policy is non-compliant.

- When the policy is in ***everyEvent*** mode and the **DelayAfterRunSeconds** exceeds the defined time value, the policy is non-compliant and the automation runs for every policy violation.

2.4. POLICY DEPLOYMENT WITH EXTERNAL TOOLS

To deploy **CertificatePolicy**, **ConfigurationPolicy**, **OperatorPolicy** resources, and Gatekeeper constraints directly to your managed cluster, you can use an external tool such as Red Hat OpenShift GitOps.

2.4.1. Deployment workflow

Your **CertificatePolicy**, **ConfigurationPolicy**, **OperatorPolicy** policies must be in the **open-cluster-management-policies** namespace or the managed cluster namespace. Policies in other namespaces are ignored and do not receive a status. If you are using a Red Hat OpenShift GitOps version with an Argo CD version earlier than 2.13, you must configure Red Hat Advanced Cluster Management for Kubernetes policy health checks

The OpenShift GitOps service account must have permission to manage Red Hat Advanced Cluster Management for Kubernetes policies. Deploy policies with OpenShift GitOps and view the policies from the *Discovered policies* table of the Governance dashboard. The policies are grouped by the policy **name** and **kind** fields.

2.4.2. Additional resources

- See Configuring policy health checks in OpenShift GitOps .
- See Creating a ClusterRole resource for Red Hat OpenShift GitOps .

CHAPTER 3. POLICY CONTROLLER ADVANCED CONFIGURATION

You can customize policy controller configurations on your managed clusters by using the **ManagedClusterAddOn** custom resources. The following **ManagedClusterAddOns** configure the policy framework, Kubernetes configuration policy controller, and the Certificate policy controller. **Required access:** Cluster administrator

- Configure the concurrency of the governance framework
- Configure the concurrency of the configuration policy controller
- Configure the rate of requests to the API server
- Configure debug log
- Governance metric
- Verify configuration changes

3.1. CONFIGURE THE CONCURRENCY OF THE GOVERNANCE FRAMEWORK

Configure the concurrency of the governance framework for each managed cluster. To change the default value of **2**, set the **policy-evaluation-concurrency** annotation with a nonzero integer within quotation marks. Then set the value on the **ManagedClusterAddOn** object name to **governance-policy-framework** in the managed cluster namespace of the hub cluster.

See the following YAML example where the concurrency is set to **2** on the managed cluster named **cluster1**:

```
apiVersion: addon.open-cluster-management.io/v1alpha1
kind: ManagedClusterAddOn
metadata:
  name: governance-policy-framework
  namespace: cluster1
  annotations:
    policy-evaluation-concurrency: "2"
spec:
  installNamespace: open-cluster-management-agent-addon
```

To set the **client-qps** and **client-burst** annotations, update the **ManagedClusterAddOn** resource and define the parameters.

See the following YAML example where the queries for each second is set to **30** and the burst is set to **45** on the managed cluster called **cluster1**:

```
apiVersion: addon.open-cluster-management.io/v1alpha1
kind: ManagedClusterAddOn
metadata:
  name: governance-policy-framework
  namespace: cluster1
  annotations:
    client-qps: "30"
```

```
    client-burst: "45"
spec:
  installNamespace: open-cluster-management-agent-addon
```

3.2. CONFIGURE THE CONCURRENCY OF THE CONFIGURATION POLICY CONTROLLER

You can configure the concurrency of the configuration policy controller for each managed cluster to change how many configuration policies it can evaluate at the same time. To change the default value of **2**, set the **policy-evaluation-concurrency** annotation with a nonzero integer within quotation marks. Then set the value on the **ManagedClusterAddOn** object name to **config-policy-controller** in the managed cluster namespace of the hub cluster.

Note: Increased concurrency values increase CPU and memory utilization on the **config-policy-controller** pod, Kubernetes API server, and OpenShift API server.

See the following YAML example where the concurrency is set to **5** on the managed cluster named **cluster1**:

```
apiVersion: addon.open-cluster-management.io/v1alpha1
kind: ManagedClusterAddOn
metadata:
  name: config-policy-controller
  namespace: cluster1
  annotations:
    policy-evaluation-concurrency: "5"
spec:
  installNamespace: open-cluster-management-agent-addon
```

3.3. CONFIGURE THE RATE OF REQUESTS TO THE API SERVER

Configure the rate of requests to the API server that the configuration policy controller makes on each managed cluster. An increased rate improves the responsiveness of the configuration policy controller, which also increases the CPU and memory utilization of the Kubernetes API server and OpenShift API server. By default, the rate of requests scales with the **policy-evaluation-concurrency** setting and is set to **30** queries for each second (QPS), with a **45** burst value, representing a higher number of requests over short periods of time.

You can configure the rate and burst by setting the **client-qps** and **client-burst** annotations with nonzero integers within quotation marks. You can set the value on the **ManagedClusterAddOn** object name to **config-policy-controller** in the managed cluster namespace of the hub cluster.

See the following YAML example where the queries for each second is set to **20** and the burst is set to **100** on the managed cluster called **cluster1**:

```
apiVersion: addon.open-cluster-management.io/v1alpha1
kind: ManagedClusterAddOn
metadata:
  name: config-policy-controller
  namespace: cluster1
  annotations:
    client-qps: "20"
```

```
    client-burst: "100"
spec:
  installNamespace: open-cluster-management-agent-addon
```

3.4. CONFIGURE DEBUG LOG

When you configure and collect debug logs for each policy controller, you can adjust the log level.

Note: Reducing the volume of debug logs means there is less information displayed from the logs.

You can reduce the debug logs emitted by the policy controllers to be display error-only bugs in the logs. To reduce the debug logs, set the debug log value to **-1** in the annotation. See what each value represents:

- **-1**: error logs only
- **0**: informative logs
- **1**: debug logs
- **2**: verbose debugging logs

To receive the second level of debugging information for the Kubernetes configuration controller, add the **log-level** annotation with the value of **2** to the **ManagedClusterAddOn** custom resource. By default, the **log-level** is set to **0**, which means you receive informative messages. View the following example:

```
apiVersion: addon.open-cluster-management.io/v1alpha1
kind: ManagedClusterAddOn
metadata:
  name: config-policy-controller
  namespace: cluster1
  annotations:
    log-level: "2"
spec:
  installNamespace: open-cluster-management-agent-addon
```

Additionally, for each **spec.object-template[]** in a **ConfigurationPolicy** resource, you can set the parameter **recordDiff** to **Log**. The difference between the **objectDefinition** and the object on the managed cluster is logged in the **config-policy-controller** pod on the managed cluster. View the following example:

This **ConfigurationPolicy** resource with **recordDiff: Log**:

```
apiVersion: policy.open-cluster-management.io/v1
kind: ConfigurationPolicy
metadata:
  name: my-config-policy
spec:
  object-templates:
  - complianceType: musthave
    recordDiff: Log
    objectDefinition:
      apiVersion: v1
      kind: ConfigMap
```

```
  metadata:
    name: my-configmap
  data:
    fieldToUpdate: "2"
```

If the **ConfigMap** resource on the cluster lists **fieldToUpdate: "1"**, then the diff appears in the **config-policy-controller** pod logs with the following information:

```
Logging the diff:
--- default/my-configmap : existing
+++ default/my-configmap : updated
@@ -2,3 +2,3 @@
 data:
-  fieldToUpdate: "1"
+  fieldToUpdate: "2"
 kind: ConfigMap
```

Important: Avoid logging the difference for a secure object. The difference is logged in plain text.

3.5. GOVERNANCE METRIC

The policy framework exposes metrics that show policy distribution and status. Use the **policy_governance_info** metric on the hub cluster to view trends and analyze any policy failures. See the following topics for an overview of metrics:

3.5.1. Metric: policy_governance_info

The OpenShift Container Platform monitoring component collects the **policy_governance_info** metric. If you enable observability, the component collects some aggregate data.

Note: If you enable observability, enter a query for the metric from the Grafana *Explore* page. When you create a policy, you are creating a *root* policy. The framework watches for root policies, **Placement** resources, and **PlacementBindings** resources to for information about where to create *propagated* policies, to distribute the policy to managed clusters.

For both root and propagated policies, a metric of **0** is recorded if the policy is compliant, **1** if it is non-compliant, and **-1** if it is in an unknown or pending state.

The **policy_governance_info** metric uses the following labels:

- **type**: The label values are **root** or **propagated**.
- **policy**: The name of the associated root policy.
- **policy_namespace**: The namespace on the hub cluster where the root policy is defined.
- **cluster_namespace**: The namespace for the cluster where the policy is distributed.

These labels and values enable queries that can show us many things happening in the cluster that might be difficult to track.

Note: If you do not need the metrics, and you have any concerns about performance or security, you can disable the metric collection. Set the **DISABLE_REPORT_METRICS** environment variable to **true** in the propagator deployment. You can also add **policy_governance_info** metric to the observability

allowlist as a custom metric. See Adding custom metrics for more details.

3.5.2. Metric: *config_policies_evaluation_duration_seconds*

The **config_policies_evaluation_duration_seconds** histogram tracks the number of seconds it takes to process all configuration policies that are ready to be evaluated on the cluster. Use the following metrics to query the histogram:

- **config_policies_evaluation_duration_seconds_bucket**: The buckets are cumulative and represent seconds with the following possible entries: 1, 3, 9, 10.5, 15, 30, 60, 90, 120, 180, 300, 450, 600, and greater.
- **config_policies_evaluation_duration_seconds_count**: The count of all events.
- **config_policies_evaluation_duration_seconds_sum**: The sum of all values.

Use the **config_policies_evaluation_duration_seconds** metric to determine if the **ConfigurationPolicy evaluationInterval** setting needs to be changed for resource intensive policies that do not need frequent evaluation. You can also increase the concurrency at the cost of higher resource utilization on the Kubernetes API server. See *Configure the concurrency* section for more details.

To receive information about the time used to evaluate configuration policies, perform a Prometheus query that resembles the following expression:

rate(config_policies_evaluation_duration_seconds_sum[10m])/rate (config_policies_evaluation_duration_seconds_count[10m]

The **config-policy-controller** pod running on managed clusters in the **open-cluster-management-agent-addon** namespace calculates the metric. The **config-policy-controller** does not send the metric to observability by default.

3.6. VERIFY CONFIGURATION CHANGES

When you apply the new configuration with the controller, the **ManifestApplied** parameter is updated in the **ManagedClusterAddOn**. That condition timestamp helps verify the configuration correctly. For example, this command can verify when the **cert-policy-controller** on the **local-cluster** was updated:

```
oc get -n local-cluster managedclusteraddon cert-policy-controller | grep -B4 'type: ManifestApplied'
```

You might receive the following output:

```
- lastTransitionTime: "2023-01-26T15:42:22Z"
  message: manifests of addon are applied successfully
  reason: AddonManifestApplied
  status: "True"
  type: ManifestApplied
```

3.7. ADDITIONAL RESOURCES

- See Kubernetes configuration policy controller
- Return to the Governance topic for more topics.

- Return to the beginning of this topic, Policy controller advanced configuration .

CHAPTER 4. SUPPORTED RED HAT ADVANCED CLUSTER MANAGEMENT FOR KUBERNETES POLICIES

View the supported policies to learn how to define rules, processes, and controls on the hub cluster when you create and manage policies in Red Hat Advanced Cluster Management for Kubernetes.

4.1. TABLE OF SAMPLE CONFIGURATION POLICIES

View the following sample configuration policies:

Table 4.1. Table list of configuration policies

Policy sample	Description
Namespace policy	Ensure consistent environment isolation and naming with Namespaces. See the Kubernetes Namespace documentation.
Pod policy	Ensure cluster workload configuration. See the Kubernetes Pod documentation.
Memory usage policy	Limit workload resource usage using Limit Ranges. See the Limit Range documentation.
Pod security policy (Deprecated)	Ensure consistent workload security. See the Kubernetes Pod security policy documentation.
Role policy Role binding policy	Manage role permissions and bindings using roles and role bindings. See the Kubernetes RBAC documentation.
Security content constraints (SCC) policy	Manage workload permissions with Security Context Constraints. See Managing Security Context Constraints documentation in the OpenShift Container Platform documentation.
ETCD encryption policy	Ensure data security with etcd encryption. See Encrypting etcd data in the OpenShift Container Platform documentation.
Compliance operator policy	Deploy the Compliance Operator to scan and enforce the compliance state of clusters leveraging OpenSCAP. See Understanding the Compliance Operator in the OpenShift Container Platform documentation.
Compliance operator E8 scan	After applying the Compliance operator policy, deploy an Essential 8 (E8) scan to check for compliance with E8 security profiles. See Understanding the Compliance Operator in the OpenShift Container Platform documentation.

Policy sample	Description
Compliance operator CIS scan	After applying the Compliance operator policy, deploy a Center for Internet Security (CIS) scan to check for compliance with CIS security profiles. See Understanding the Compliance Operator in the OpenShift Container Platform documentation.
Image vulnerability policy	Deploy the Container Security Operator and detect known image vulnerabilities in pods running on the cluster. See the Container Security Operator GitHub repository.
Gatekeeper operator deployment	Gatekeeper is an admission webhook that enforces custom resource definition-based policies that are run by the Open Policy Agent (OPA) policy engine. See the Gatekeeper documentation. The Gatekeeper operator is available for installing Gatekeeper. For more information, see the Gatekeeper operator overview.
Gatekeeper compliance policy	After deploying Gatekeeper to the clusters, deploy this sample Gatekeeper policy that ensures namespaces that are created on the cluster are labeled as specified. For more information, see Integrating Gatekeeper constraints and constraint templates.
Red Hat OpenShift Platform Plus policy set	Red Hat OpenShift Platform Plus is a hybrid-cloud suite of products to securely build, deploy, run, and manage applications for multiple infrastructures. You can deploy Red Hat OpenShift Platform Plus to managed clusters using **PolicySets** delivered through a Red Hat Advanced Cluster Management application. For details on OpenShift Platform Plus, see the documentation for OpenShift Platform Plus.

Red Hat OpenShift Container Platform 4.x also supports the Red Hat Advanced Cluster Management configuration policies.

View the following policy documentation to learn how policies are applied:

- Namespace policy
- Pod policy
- Memory usage policy
- Pod security policy
- Role policy
- Role binding policy

- Security context constraints policy
- ETCD encryption policy
- Compliance operator policy
- E8 scan policy
- OpenShift CIS scan policy
- Image vulnerability policy
- Gatekeeper operator overview
- Integrating Gatekeeper constraints and constraint templates
- Red Hat OpenShift Platform Plus policy set

Refer to Governance for more topics.

4.2. NAMESPACE POLICY

The Kubernetes configuration policy controller monitors the status of your namespace policy. Apply the namespace policy to define specific rules for your namespace.

Learn more details about the namespace policy structure in the following sections:

- Namespace policy YAML structure
- Namespace policy table
- Namespace policy sample

4.2.1. Namespace policy YAML structure

```
apiVersion: policy.open-cluster-management.io/v1
kind: Policy
metadata:
  name:
  namespace:
  annotations:
    policy.open-cluster-management.io/standards:
    policy.open-cluster-management.io/categories:
    policy.open-cluster-management.io/controls:
    policy.open-cluster-management.io/description:
spec:
  remediationAction:
  disabled:
  policy-templates:
    - objectDefinition:
        apiVersion: policy.open-cluster-management.io/v1
        kind: ConfigurationPolicy
        metadata:
          name:
        spec:
          remediationAction:
```

```
    severity:
object-templates:
  - complianceType:
    objectDefinition:
      kind: Namespace
      apiVersion: v1
      metadata:
        name:
      ...
```

4.2.2. Namespace policy YAML table

Field	Optional or required	Description
apiVersion	Required	Set the value to **policy.open-cluster-management.io/v1**.
kind	Required	Set the value to **Policy** to indicate the type of policy.
metadata.name	Required	The name for identifying the policy resource.
metadata.namespace	Required	The namespace of the policy.
spec.remediationAction	Optional	Specifies the remediation of your policy. The parameter values are **enforce** and **inform**. This value is optional because it overrides any values provided in **spec.policy-templates**.
spec.disabled	Required	Set the value to **true** or **false**. The **disabled** parameter provides the ability to enable and disable your policies.
spec.policy-templates[].objectDefinition	Required	Used to list configuration policies containing Kubernetes objects that must be evaluated or applied to the managed clusters.

4.2.3. Namespace policy sample

See **policy-namespace.yaml** to view the policy sample.

See Managing security policies for more details. Refer to Hub cluster policy framework documentation, and to the Kubernetes configuration policy controller to learn about other configuration policies.

4.3. POD POLICY

The Kubernetes configuration policy controller monitors the status of your pod policies. Apply the pod policy to define the container rules for your pods. A pod must exist in your cluster to use this information.

Learn more details about the pod policy structure in the following sections:

- Pod policy YAML structure
- Pod policy table
- Pod policy sample

4.3.1. Pod policy YAML structure

```
apiVersion: policy.open-cluster-management.io/v1
kind: Policy
metadata:
  name:
  namespace:
  annotations:
    policy.open-cluster-management.io/standards:
    policy.open-cluster-management.io/categories:
    policy.open-cluster-management.io/controls:
    policy.open-cluster-management.io/description:
spec:
  remediationAction:
  disabled:
  policy-templates:
    - objectDefinition:
        apiVersion: policy.open-cluster-management.io/v1
        kind: ConfigurationPolicy
        metadata:
          name:
        spec:
          remediationAction:
          severity:
          namespaceSelector:
            exclude:
            include:
            matchLabels:
            matchExpressions:
          object-templates:
            - complianceType:
              objectDefinition:
                apiVersion: v1
                kind: Pod
                metadata:
                  name:
                spec:
                  containers:
                  - image:
                    name:
                ...
```

4.3.2. Pod policy table

Table 4.2. Parameter table

Field	Optional or required	Description
apiVersion	Required	Set the value to **policy.open-cluster-management.io/v1**.
kind	Required	Set the value to **Policy** to indicate the type of policy.
metadata.name	Required	The name for identifying the policy resource.
metadata.namespace	Required	The namespace of the policy.
spec.remediationAction	Optional	Specifies the remediation of your policy. The parameter values are **enforce** and **inform**. This value is optional because the value overrides any values provided in **spec.policy-templates**.
spec.disabled	Required	Set the value to **true** or **false**. The **disabled** parameter provides the ability to enable and disable your policies.
spec.policy-templates[].objectDefinition	Required	Used to list configuration policies containing Kubernetes objects that must be evaluated or applied to the managed clusters.

4.3.3. Pod policy sample

See **policy-pod.yaml** to view the policy sample.

Refer to Kubernetes configuration policy controller to view other configuration policies that are monitored by the configuration controller, and see the Hub cluster policy framework to see a full description of the policy YAML structure and additional fields. Return to Creating configuration policies documentation to manage other policies.

4.4. MEMORY USAGE POLICY

The Kubernetes configuration policy controller monitors the status of the memory usage policy. Use the memory usage policy to limit or restrict your memory and compute usage. For more information, see *Limit Ranges* in the Kubernetes documentation.

Learn more details about the memory usage policy structure in the following sections:

- Memory usage policy YAML structure

- Memory usage policy table
- Memory usage policy sample

4.4.1. Memory usage policy YAML structure

Your memory usage policy might resemble the following YAML file:

```
apiVersion: policy.open-cluster-management.io/v1
kind: Policy
metadata:
  name:
  namespace:
  annotations:
    policy.open-cluster-management.io/standards:
    policy.open-cluster-management.io/categories:
    policy.open-cluster-management.io/controls:
    policy.open-cluster-management.io/description:
spec:
  remediationAction:
  disabled:
  policy-templates:
    - objectDefinition:
        apiVersion: policy.open-cluster-management.io/v1
        kind: ConfigurationPolicy
        metadata:
          name:
        spec:
          remediationAction:
          severity:
          namespaceSelector:
            exclude:
            include:
            matchLabels:
            matchExpressions:
          object-templates:
            - complianceType: mustonlyhave
              objectDefinition:
                apiVersion: v1
                kind: LimitRange
                metadata:
                  name:
                spec:
                  limits:
                  - default:
                      memory:
                    defaultRequest:
                      memory:
                    type:
        ...
```

4.4.2. Memory usage policy table

Table 4.3. Parameter table

Field	Optional or required	Description
apiVersion	Required	Set the value to **policy.open-cluster-management.io/v1**.
kind	Required	Set the value to **Policy** to indicate the type of policy.
metadata.name	Required	The name for identifying the policy resource.
metadata.namespace	Required	The namespace of the policy.
spec.remediationAction	Optional	Specifies the remediation of your policy. The parameter values are **enforce** and **inform**. This value is optional because the value overrides any values provided in **spec.policy-templates**.
spec.disabled	Required	Set the value to **true** or **false**. The **disabled** parameter provides the ability to enable and disable your policies.
spec.policy-templates[].objectDefinition	Required	Used to list configuration policies containing Kubernetes objects that must be evaluated or applied to the managed clusters.

4.4.3. Memory usage policy sample

See the **policy-limitmemory.yaml** to view a sample of the policy. See Managing security policies for more details. Refer to the Hub cluster policy framework documentation, and to Kubernetes configuration policy controller to view other configuration policies that are monitored by the controller.

4.5. POD SECURITY POLICY (DEPRECATED)

The Kubernetes configuration policy controller monitors the status of the pod security policy. Apply a pod security policy to secure pods and containers.

Learn more details about the pod security policy structure in the following sections:

- Pod security policy YAML structure
- Pod security policy table
- Pod security policy sample

4.5.1. Pod security policy YAML structure

```
apiVersion: policy.open-cluster-management.io/v1
kind: Policy
metadata:
  name:
  namespace:
  annotations:
    policy.open-cluster-management.io/standards:
    policy.open-cluster-management.io/categories:
    policy.open-cluster-management.io/controls:
    policy.open-cluster-management.io/description:
spec:
  remediationAction:
  disabled:
  policy-templates:
    - objectDefinition:
        apiVersion: policy.open-cluster-management.io/v1
        kind: ConfigurationPolicy
        metadata:
          name:
        spec:
          remediationAction:
          severity:
          namespaceSelector:
            exclude:
            include:
            matchLabels:
            matchExpressions:
          object-templates:
            - complianceType:
              objectDefinition:
                apiVersion: policy/v1beta1
                kind: PodSecurityPolicy
                metadata:
                  name:
                  annotations:
                    seccomp.security.alpha.kubernetes.io/allowedProfileNames:
                spec:
                  privileged:
                  allowPrivilegeEscalation:
                  allowedCapabilities:
                  volumes:
                  hostNetwork:
                  hostPorts:
                  hostIPC:
                  hostPID:
                  runAsUser:
                  seLinux:
                  supplementalGroups:
                  fsGroup:
                ...
```

4.5.2. Pod security policy table

Table 4.4. Parameter table

Field	Optional or required	Description
apiVersion	Required	Set the value to **policy.open-cluster-management.io/v1**.
kind	Required	Set the value to **Policy** to indicate the type of policy.
metadata.name	Required	The name for identifying the policy resource.
metadata.namespace	Required	The namespace of the policy.
spec.remediationAction	Optional	Specifies the remediation of your policy. The parameter values are **enforce** and **inform**. This value is optional because the value overrides any values provided in **spec.policy-templates**.
spec.disabled	Required	Set the value to **true** or **false**. The **disabled** parameter provides the ability to enable and disable your policies.
spec.policy-templates[].objectDefinition	Required	Used to list configuration policies containing Kubernetes objects that must be evaluated or applied to the managed clusters.

4.5.3. Pod security policy sample

The support of pod security policies is removed from OpenShift Container Platform and from Kubernetes v1.25 and later. If you apply a **PodSecurityPolicy** resource, you might receive the following non-compliant message:

> violation - couldn't find mapping resource with kind PodSecurityPolicy, please check if you have CRD deployed

- For more information including the deprecation notice, see *Pod Security Policies* in the Kubernetes documentation.
- See **policy-psp.yaml** to view the sample policy. View Creating configuration policies for more information.
- Refer to the Hub cluster policy framework documentation for a full description of the policy YAML structure, and Kubernetes configuration policy controller to view other configuration policies that are monitored by the controller.

4.6. ROLE POLICY

The Kubernetes configuration policy controller monitors the status of role policies. Define roles in the **object-template** to set rules and permissions for specific roles in your cluster.

Learn more details about the role policy structure in the following sections:

- Role policy YAML structure
- Role policy table
- Role policy sample

4.6.1. Role policy YAML structure

```
apiVersion: policy.open-cluster-management.io/v1
kind: Policy
metadata:
  name:
  namespace:
  annotations:
    policy.open-cluster-management.io/standards:
    policy.open-cluster-management.io/categories:
    policy.open-cluster-management.io/controls:
    policy.open-cluster-management.io/description:
spec:
  remediationAction:
  disabled:
  policy-templates:
    - objectDefinition:
        apiVersion: policy.open-cluster-management.io/v1
        kind: ConfigurationPolicy
        metadata:
          name:
        spec:
          remediationAction:
          severity:
          namespaceSelector:
            exclude:
            include:
            matchLabels:
            matchExpressions:
          object-templates:
            - complianceType:
              objectDefinition:
                apiVersion: rbac.authorization.k8s.io/v1
                kind: Role
                metadata:
                  name:
                rules:
                  - apiGroups:
                    resources:
                    verbs:
                ...
```

4.6.2. Role policy table

Table 4.5. Parameter table

Field	Optional or required	Description
apiVersion	Required	Set the value to **policy.open-cluster-management.io/v1**.
kind	Required	Set the value to **Policy** to indicate the type of policy.
metadata.name	Required	The name for identifying the policy resource.
metadata.namespace	Required	The namespace of the policy.
spec.remediationAction	Optional	Specifies the remediation of your policy. The parameter values are **enforce** and **inform**. This value is optional because the value overrides any values provided in **spec.policy-templates**.
spec.disabled	Required	Set the value to **true** or **false**. The **disabled** parameter provides the ability to enable and disable your policies.
spec.policy-templates[].objectDefinition	Required	Used to list configuration policies containing Kubernetes objects that must be evaluated or applied to the managed clusters.

4.6.3. Role policy sample

Apply a role policy to set rules and permissions for specific roles in your cluster. For more information on roles, see Role-based access control. View a sample of a role policy, see **policy-role.yaml**.

To learn how to manage role policies, refer to Creating configuration policies for more information. See the Kubernetes configuration policy controller to view other configuration policies that are monitored the controller.

4.7. ROLE BINDING POLICY

The Kubernetes configuration policy controller monitors the status of your role binding policy. Apply a role binding policy to bind a policy to a namespace in your managed cluster.

Learn more details about the namespace policy structure in the following sections:

- Role binding policy YAML structure
- Role binding policy table

- Role binding policy sample

4.7.1. Role binding policy YAML structure

```
apiVersion: policy.open-cluster-management.io/v1
kind: Policy
metadata:
  name:
  namespace:
  annotations:
    policy.open-cluster-management.io/standards:
    policy.open-cluster-management.io/categories:
    policy.open-cluster-management.io/controls:
    policy.open-cluster-management.io/description:
spec:
  remediationAction:
  disabled:
  policy-templates:
    - objectDefinition:
        apiVersion: policy.open-cluster-management.io/v1
        kind: ConfigurationPolicy
        metadata:
          name:
        spec:
          remediationAction:
          severity:
          namespaceSelector:
            exclude:
            include:
            matchLabels:
            matchExpressions:
          object-templates:
            - complianceType:
              objectDefinition:
                kind: RoleBinding # role binding must exist
                apiVersion: rbac.authorization.k8s.io/v1
                metadata:
                  name:
                subjects:
                - kind:
                  name:
                  apiGroup:
                roleRef:
                  kind:
                  name:
                  apiGroup:
                ...
```

4.7.2. Role binding policy table

Field	Optional or required	Description

Field	Optional or required	Description
apiVersion	Required	Set the value to **policy.open-cluster-management.io/v1**.
kind	Required	Set the value to **Policy** to indicate the type of policy.
metadata.name	Required	The name for identifying the policy resource.
metadata.namespace	Required	The namespace of the policy.
spec.remediationAction	Optional	Specifies the remediation of your policy. The parameter values are **enforce** and **inform**. This value is optional since it overrides any values provided in **spec.policy-templates**.
spec.disabled	Required	Set the value to **true** or **false**. The **disabled** parameter provides the ability to enable and disable your policies.
spec.policy-templates[].objectDefinition	Required	Used to list configuration policies containing Kubernetes objects that must be evaluated or applied to the managed clusters.

4.7.3. Role binding policy sample

See **policy-rolebinding.yaml** to view the policy sample. For a full description of the policy YAML structure and additional fields, see the Hub cluster policy framework. Refer to Kubernetes configuration policy controller documentation to learn about other configuration policies.

4.8. SECURITY CONTEXT CONSTRAINTS POLICY

The Kubernetes configuration policy controller monitors the status of your Security Context Constraints (SCC) policy. Apply an Security Context Constraints (SCC) policy to control permissions for pods by defining conditions in the policy.

Learn more details about SCC policies in the following sections:

- SCC policy YAML structure
- SCC policy table
- SCC policy sample

4.8.1. SCC policy YAML structure

```
apiVersion: policy.open-cluster-management.io/v1
kind: Policy
metadata:
  name:
  namespace:
  annotations:
    policy.open-cluster-management.io/standards:
    policy.open-cluster-management.io/categories:
    policy.open-cluster-management.io/controls:
    policy.open-cluster-management.io/description:
spec:
  remediationAction:
  disabled:
  policy-templates:
    - objectDefinition:
        apiVersion: policy.open-cluster-management.io/v1
        kind: ConfigurationPolicy
        metadata:
          name:
        spec:
          remediationAction:
          severity:
          namespaceSelector:
            exclude:
            include:
            matchLabels:
            matchExpressions:
          object-templates:
            - complianceType:
              objectDefinition:
                apiVersion: security.openshift.io/v1
                kind: SecurityContextConstraints
                metadata:
                  name:
                allowHostDirVolumePlugin:
                allowHostIPC:
                allowHostNetwork:
                allowHostPID:
                allowHostPorts:
                allowPrivilegeEscalation:
                allowPrivilegedContainer:
                fsGroup:
                readOnlyRootFilesystem:
                requiredDropCapabilities:
                runAsUser:
                seLinuxContext:
                supplementalGroups:
                users:
                volumes:
                ...
```

4.8.2. SCC policy table

Field	Optional or required	Description
apiVersion	Required	Set the value to **policy.open-cluster-management.io/v1**.
kind	Required	Set the value to **Policy** to indicate the type of policy.
metadata.name	Required	The name for identifying the policy resource.
metadata.namespace	Required	The namespace of the policy.
spec.remediationAction	Optional	Specifies the remediation of your policy. The parameter values are **enforce** and **inform**. This value is optional since it overrides any values provided in **spec.policy-templates**.
spec.disabled	Required	Set the value to **true** or **false**. The **disabled** parameter provides the ability to enable and disable your policies.
spec.policy-templates[].objectDefinition	Required	Used to list configuration policies containing Kubernetes objects that must be evaluated or applied to the managed clusters.

For explanations on the contents of a SCC policy, see Managing Security Context Constraints from the OpenShift Container Platform documentation.

4.8.3. SCC policy sample

Apply a Security context constraints (SCC) policy to control permissions for pods by defining conditions in the policy. For more information, see Managing Security Context Constraints.

See **policy-scc.yaml** to view the policy sample. For a full description of the policy YAML structure and additional fields, see the Hub cluster policy framework documentation. Refer to Kubernetes configuration policy controller documentation to learn about other configuration policies.

4.9. ETCD ENCRYPTION POLICY

Apply the **etcd-encryption** policy to detect, or enable encryption of sensitive data in the ETCD data-store. The Kubernetes configuration policy controller monitors the status of the **etcd-encryption** policy. For more information, see Encrypting etcd data in the OpenShift Container Platform documentation.
Note: The ETCD encryption policy only supports Red Hat OpenShift Container Platform 4 and later.

Learn more details about the **etcd-encryption** policy structure in the following sections:

- ETCD encryption policy YAML structure
- ETCD encryption policy table
- ETCD encryption policy sample

4.9.1. ETCD encryption policy YAML structure

Your **etcd-encryption** policy might resemble the following YAML file:

```
apiVersion: policy.open-cluster-management.io/v1
kind: Policy
metadata:
  name:
  namespace:
  annotations:
    policy.open-cluster-management.io/standards:
    policy.open-cluster-management.io/categories:
    policy.open-cluster-management.io/controls:
    policy.open-cluster-management.io/description:
spec:
  remediationAction:
  disabled:
  policy-templates:
    - objectDefinition:
        apiVersion: policy.open-cluster-management.io/v1
        kind: ConfigurationPolicy
        metadata:
          name:
        spec:
          remediationAction:
          severity:
          object-templates:
            - complianceType:
              objectDefinition:
                apiVersion: config.openshift.io/v1
                kind: APIServer
                metadata:
                  name:
                spec:
                  encryption:
                ...
```

4.9.2. ETCD encryption policy table

Table 4.6. Parameter table

Field	Optional or required	Description
apiVersion	Required	Set the value to **policy.open-cluster-management.io/v1**.

Field	Optional or required	Description
kind	Required	Set the value to **Policy** to indicate the type of policy.
metadata.name	Required	The name for identifying the policy resource.
metadata.namespace	Required	The namespace of the policy.
spec.remediationAction	Optional	Specifies the remediation of your policy. The parameter values are **enforce** and **inform**. This value is optional because it overrides any values provided in **spec.policy-templates**.
spec.disabled	Required	Set the value to **true** or **false**. The **disabled** parameter provides the ability to enable and disable your policies.
spec.policy-templates[].objectDefinition	Required	Used to list configuration policies containing Kubernetes objects that must be evaluated or applied to the managed clusters.

4.9.3. ETCD encryption policy sample

See **policy-etcdencryption.yaml** for the policy sample. See the Hub cluster policy framework documentation and the Kubernetes configuration policy controller to view additional details on policy and configuration policy fields.

4.10. COMPLIANCE OPERATOR POLICY

You can use the Compliance Operator to automate the inspection of numerous technical implementations and compare those against certain aspects of industry standards, benchmarks, and baselines. The Compliance Operator is not an auditor. To be compliant or certified with these various standards, you need to engage an authorized auditor such as a Qualified Security Assessor (QSA), Joint Authorization Board (JAB), or other industry recognized regulatory authority to assess your environment.

Recommendations that are generated from the Compliance Operator are based on generally available information and practices regarding such standards, and might assist you with remediations, but actual compliance is your responsibility. Work with an authorized auditor to achieve compliance with a standard.

For the latest updates, see the Compliance Operator release notes.

4.10.1. Compliance Operator policy overview

You can install the Compliance Operator on your managed cluster by using the Compliance Operator

policy. The Compliance operator policy is created as a Kubernetes configuration policy in Red Hat Advanced Cluster Management. OpenShift Container Platform supports the compliance operator policy.

Note: The Compliance operator policy relies on the OpenShift Container Platform Compliance Operator, which is not supported on the IBM Power or IBM Z architectures. See Understanding the Compliance Operator in the OpenShift Container Platform documentation for more information about the Compliance Operator.

4.10.2. Compliance operator resources

When you create a compliance operator policy, the following resources are created:

- A compliance operator namespace (**openshift-compliance**) for the operator installation:

```
apiVersion: policy.open-cluster-management.io/v1
kind: ConfigurationPolicy
metadata:
  name: comp-operator-ns
spec:
  remediationAction: inform # will be overridden by remediationAction in parent policy
  severity: high
  object-templates:
    - complianceType: musthave
      objectDefinition:
        apiVersion: v1
        kind: Namespace
        metadata:
          name: openshift-compliance
```

- An operator group (**compliance-operator**) to specify the target namespace:

```
apiVersion: policy.open-cluster-management.io/v1
kind: ConfigurationPolicy
metadata:
  name: comp-operator-operator-group
spec:
  remediationAction: inform # will be overridden by remediationAction in parent policy
  severity: high
  object-templates:
    - complianceType: musthave
      objectDefinition:
        apiVersion: operators.coreos.com/v1
        kind: OperatorGroup
        metadata:
          name: compliance-operator
          namespace: openshift-compliance
        spec:
          targetNamespaces:
            - openshift-compliance
```

- A subscription (**comp-operator-subscription**) to reference the name and channel. The subscription pulls the profile, as a container, that it supports. See the following sample, with the current version replacing **4.x**:

```
apiVersion: policy.open-cluster-management.io/v1
kind: ConfigurationPolicy
metadata:
  name: comp-operator-subscription
spec:
  remediationAction: inform  # will be overridden by remediationAction in parent policy
  severity: high
  object-templates:
    - complianceType: musthave
      objectDefinition:
        apiVersion: operators.coreos.com/v1alpha1
        kind: Subscription
        metadata:
          name: compliance-operator
          namespace: openshift-compliance
        spec:
          channel: "4.x"
          installPlanApproval: Automatic
          name: compliance-operator
          source: redhat-operators
          sourceNamespace: openshift-marketplace
```

After you install the compliance operator policy, the following pods are created: **compliance-operator**, **ocp4**, and **rhcos4**. See a sample of the **policy-compliance-operator-install.yaml**.

4.10.3. Additional resources

- For more information, see Managing the Compliance Operator in the OpenShift Container Platform documentation for more details.
- You can also create and apply the E8 scan policy and OpenShift CIS scan policy, after you have installed the compliance operator. For more information, see E8 scan policy and OpenShift CIS scan policy.
- To learn about managing compliance operator policies, see Managing security policies for more details. Refer to Kubernetes configuration policy controller for more topics about configuration policies.

4.11. E8 SCAN POLICY

An Essential 8 (E8) scan policy deploys a scan that checks the master and worker nodes for compliance with the E8 security profiles. You must install the compliance operator to apply the E8 scan policy.

The E8 scan policy is created as a Kubernetes configuration policy in Red Hat Advanced Cluster Management. OpenShift Container Platform supports the E8 scan policy. For more information, see Managing the Compliance Operator in the OpenShift Container Platform documentation for more details.

4.11.1. E8 scan policy resources

When you create an E8 scan policy the following resources are created:

- A **ScanSettingBinding** resource (**e8**) to identify which profiles to scan:

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

```
kind: ConfigurationPolicy
metadata:
  name: compliance-suite-e8
spec:
  remediationAction: inform
  severity: high
  object-templates:
    - complianceType: musthave # this template checks if scan has completed by checking the
status field
      objectDefinition:
        apiVersion: compliance.openshift.io/v1alpha1
        kind: ScanSettingBinding
        metadata:
          name: e8
          namespace: openshift-compliance
        profiles:
        - apiGroup: compliance.openshift.io/v1alpha1
          kind: Profile
          name: ocp4-e8
        - apiGroup: compliance.openshift.io/v1alpha1
          kind: Profile
          name: rhcos4-e8
        settingsRef:
          apiGroup: compliance.openshift.io/v1alpha1
          kind: ScanSetting
          name: default
```

- A **ComplianceSuite** resource (**compliance-suite-e8**) to verify if the scan is complete by checking the **status** field:

```
apiVersion: policy.open-cluster-management.io/v1
kind: ConfigurationPolicy
metadata:
  name: compliance-suite-e8
spec:
  remediationAction: inform
  severity: high
  object-templates:
    - complianceType: musthave # this template checks if scan has completed by checking the
status field
      objectDefinition:
        apiVersion: compliance.openshift.io/v1alpha1
        kind: ComplianceSuite
        metadata:
          name: e8
          namespace: openshift-compliance
        status:
          phase: DONE
```

- A **ComplianceCheckResult** resource (**compliance-suite-e8-results**) which reports the results of the scan suite by checking the **ComplianceCheckResult** custom resources (CR):

```
apiVersion: policy.open-cluster-management.io/v1
kind: ConfigurationPolicy
metadata:
```

```
  name: compliance-suite-e8-results
spec:
  remediationAction: inform
  severity: high
  object-templates:
    - complianceType: mustnothave # this template reports the results for scan suite: e8 by
looking at ComplianceCheckResult CRs
      objectDefinition:
        apiVersion: compliance.openshift.io/v1alpha1
        kind: ComplianceCheckResult
        metadata:
          namespace: openshift-compliance
          labels:
            compliance.openshift.io/check-status: FAIL
            compliance.openshift.io/suite: e8
```

Note: Automatic remediation is supported. Set the remediation action to **enforce** to create **ScanSettingBinding** resource.

See a sample of the **policy-compliance-operator-e8-scan.yaml**. See Managing security policies for more information. **Note:** After your E8 policy is deleted, it is removed from your target cluster or clusters.

4.12. OPENSHIFT CIS SCAN POLICY

An OpenShift CIS scan policy deploys a scan that checks the master and worker nodes for compliance with the OpenShift CIS security benchmark. You must install the compliance operator to apply the OpenShift CIS policy.

The OpenShift CIS scan policy is created as a Kubernetes configuration policy in Red Hat Advanced Cluster Management. OpenShift Container Platform supports the OpenShift CIS scan policy. For more information, see Understanding the Compliance Operator in the OpenShift Container Platform documentation for more details.

4.12.1. OpenShift CIS resources

When you create an OpenShift CIS scan policy the following resources are created:

- A **ScanSettingBinding** resource (**cis**) to identify which profiles to scan:

```
apiVersion: policy.open-cluster-management.io/v1
kind: ConfigurationPolicy
metadata:
  name: compliance-cis-scan
spec:
  remediationAction: inform
  severity: high
  object-templates:
    - complianceType: musthave # this template creates ScanSettingBinding:cis
      objectDefinition:
        apiVersion: compliance.openshift.io/v1alpha1
        kind: ScanSettingBinding
        metadata:
          name: cis
          namespace: openshift-compliance
```

```
      profiles:
      - apiGroup: compliance.openshift.io/v1alpha1
        kind: Profile
        name: ocp4-cis
      - apiGroup: compliance.openshift.io/v1alpha1
        kind: Profile
        name: ocp4-cis-node
      settingsRef:
        apiGroup: compliance.openshift.io/v1alpha1
        kind: ScanSetting
        name: default
```

- A **ComplianceSuite** resource (**compliance-suite-cis**) to verify if the scan is complete by checking the **status** field:

```
apiVersion: policy.open-cluster-management.io/v1
kind: ConfigurationPolicy
metadata:
  name: compliance-suite-cis
spec:
  remediationAction: inform
  severity: high
  object-templates:
    - complianceType: musthave # this template checks if scan has completed by checking the
status field
      objectDefinition:
        apiVersion: compliance.openshift.io/v1alpha1
        kind: ComplianceSuite
        metadata:
          name: cis
          namespace: openshift-compliance
        status:
          phase: DONE
```

- A **ComplianceCheckResult** resource (**compliance-suite-cis-results**) which reports the results of the scan suite by checking the **ComplianceCheckResult** custom resources (CR):

```
apiVersion: policy.open-cluster-management.io/v1
kind: ConfigurationPolicy
metadata:
  name: compliance-suite-cis-results
spec:
  remediationAction: inform
  severity: high
  object-templates:
    - complianceType: mustnothave # this template reports the results for scan suite: cis by
looking at ComplianceCheckResult CRs
      objectDefinition:
        apiVersion: compliance.openshift.io/v1alpha1
        kind: ComplianceCheckResult
        metadata:
          namespace: openshift-compliance
          labels:
            compliance.openshift.io/check-status: FAIL
            compliance.openshift.io/suite: cis
```

See a sample of the **policy-compliance-operator-cis-scan.yaml** file. For more information on creating policies, see Managing security policies.

4.13. IMAGE VULNERABILITY POLICY

Apply the image vulnerability policy to detect if container images have vulnerabilities by leveraging the Container Security Operator. The policy installs the Container Security Operator on your managed cluster if it is not installed.

The image vulnerability policy is checked by the Kubernetes configuration policy controller. For more information about the Security Operator, see the *Container Security Operator* from the Quay repository.

Notes:

- Image vulnerability policy is not functional during a disconnected installation.
- The Image vulnerability policy is not supported on the IBM Power and IBM Z architectures. It relies on the Quay Container Security Operator. There are no **ppc64le** or **s390x** images in the container-security-operator registry.

View the following sections to learn more:

- Image vulnerability policy YAML structure
- Image vulnerability policy sample

4.13.1. Image vulnerability policy YAML structure

When you create the container security operator policy, it involves the following policies:

- A policy that creates the subscription (**container-security-operator**) to reference the name and channel. This configuration policy must have **spec.remediationAction** set to **enforce** to create the resources. The subscription pulls the profile, as a container, that the subscription supports. View the following example:

```
apiVersion: policy.open-cluster-management.io/v1
kind: ConfigurationPolicy
metadata:
  name: policy-imagemanifestvuln-example-sub
spec:
  remediationAction: enforce  # will be overridden by remediationAction in parent policy
  severity: high
  object-templates:
    - complianceType: musthave
      objectDefinition:
        apiVersion: operators.coreos.com/v1alpha1
        kind: Subscription
        metadata:
          name: container-security-operator
          namespace: openshift-operators
        spec:
          # channel: quay-v3.3 # specify a specific channel if desired
          installPlanApproval: Automatic
```

```
      name: container-security-operator
      source: redhat-operators
      sourceNamespace: openshift-marketplace
```

- An **inform** configuration policy to audit the **ClusterServiceVersion** to ensure that the container security operator installation succeeded. View the following example:

```
apiVersion: policy.open-cluster-management.io/v1
kind: ConfigurationPolicy
metadata:
  name: policy-imagemanifestvuln-status
spec:
  remediationAction: inform  # will be overridden by remediationAction in parent policy
  severity: high
  object-templates:
    - complianceType: musthave
      objectDefinition:
        apiVersion: operators.coreos.com/v1alpha1
        kind: ClusterServiceVersion
        metadata:
          namespace: openshift-operators
        spec:
          displayName: Red Hat Quay Container Security Operator
        status:
          phase: Succeeded   # check the CSV status to determine if operator is running or not
```

- An **inform** configuration policy to audit whether any **ImageManifestVuln** objects were created by the image vulnerability scans. View the following example:

```
apiVersion: policy.open-cluster-management.io/v1
kind: ConfigurationPolicy
metadata:
  name: policy-imagemanifestvuln-example-imv
spec:
  remediationAction: inform  # will be overridden by remediationAction in parent policy
  severity: high
  namespaceSelector:
    exclude: ["kube-*"]
    include: ["*"]
  object-templates:
    - complianceType: mustnothave # mustnothave any ImageManifestVuln object
      objectDefinition:
        apiVersion: secscan.quay.redhat.com/v1alpha1
        kind: ImageManifestVuln # checking for a Kind
```

4.13.2. Image vulnerability policy sample

See **policy-imagemanifestvuln.yaml**. See Managing security policies for more information. Refer to Kubernetes configuration policy controller to view other configuration policies that are monitored by the configuration controller.

4.14. RED HAT OPENSHIFT PLATFORM PLUS POLICY SET

Configure and apply the OpenShift Platform Plus policy set (**openshift-plus**) to install Red Hat OpenShift Platform Plus.

The OpenShift Platform Plus policy set contains two **PolicySets** that are deployed. The OpenShift Plus policy set applies multiple policies that are set to install OpenShift Platform Plus products. The Red Hat Advanced Cluster Security secured cluster services and the Compliance Operator are deployed onto all of your OpenShift Container Platform managed clusters.

4.14.1. Prerequisites

- Install Red Hat OpenShift Container Platform on Amazon Web Services (AWS) environment.
- Install Red Hat Advanced Cluster Management for Kubernetes.
- Install the Policy Generator Kustomize plugin. See the Policy Generator documentation for more information.

4.14.2. OpenShift Platform Plus policy set components

When you apply the policy set to the hub cluster, the following OpenShift Platform Plus components are installed:

Table 4.7. Component table

Component	Policy	Description
Red Hat Advanced Cluster Security	**policy-acs-central-ca-bundle**	Policy used to install the central server onto the Red Hat Advanced Cluster Management for Kubernetes hub cluster and the managed clusters.
	policy-acs-central-status	Deployments to receive Red Hat Advanced Cluster Security status.
	policy-acs-operator-central	Configuration for the Red Hat Advanced Cluster Security central operator.
	policy-acs-sync-resources	Policy used to verify that the Red Hat Advanced Cluster Security resources are created.
OpenShift Container Platform	**policy-advanced-managed-cluster-status**	The managed hub cluster. Manager of the managed cluster.
Compliance operator	**policy-compliance-operator-install**	Policy used to install the Compliance operator.
Red Hat Quay	**policy-config-quay**	Configuration policy for Red Hat Quay.

Component	Policy	Description
	policy-install-quay	Policy used to install Red Hat Quay.
	policy-quay-status	Installed onto the Red Hat Advanced Cluster Management hub cluster.
Red Hat Advanced Cluster Management	**policy-ocm-observability**	Sets up the Red Hat Advanced Cluster Management observability service.
Red Hat OpenShift Data Platform	**policy-odf**	Available storage for the hub cluster components that is used by Red Hat Advanced Cluster Management observability and Quay.
	policy-odf-status	Policy used to configure the Red Hat OpenShift Data Platform status.

4.14.3. Additional resources

- See Installing Red Hat OpenShift Platform Plus by using a policy set .
- Return to Policy set controller.
- View the **openshift-plus** policy set sample for all of the policies included in the policy set.

4.15. MANAGING SECURITY POLICIES

Create a security policy to report and validate your cluster compliance based on your specified security standards, categories, and controls.

View the following sections:

- Creating a security policy
- Updating security policies
- Deleting a security policy
- Cleaning up resources that are created by policies
- Policy command line interface

4.15.1. Creating a security policy

You can create a security policy from the command line interface (CLI) or from the console.

Required access: Cluster administrator

Important: * You must define a placement and placement binding to apply your policy to a specific cluster. The **PlacementBinding** resource binds the placement. Enter a valid value for the cluster *Label selector* field to define a **Placement** and **PlacementBinding** resource. * In order to use a **Placement** resource, a **ManagedClusterSet** resource must be bound to the namespace of the **Placement** resource with a **ManagedClusterSetBinding** resource. Refer to Creating a *ManagedClusterSetBinding* resource for additional details.

4.15.1.1. Creating a security policy from the command line interface

Complete the following steps to create a policy from the command line interface (CLI):

1. Create a policy by running the following command:

```
oc create -f policy.yaml -n <policy-namespace>
```

2. Define the template that the policy uses. Edit your YAML file by adding a **policy-templates** field to define a template. Your policy might resemble the following YAML file:

```
apiVersion: policy.open-cluster-management.io/v1
kind: Policy
metadata:
  name: policy1
spec:
  remediationAction: "enforce" # or inform
  disabled: false # or true
  namespaceSelector:
    include:
    - "default"
    - "my-namespace"
  policy-templates:
    - objectDefinition:
        apiVersion: policy.open-cluster-management.io/v1
        kind: ConfigurationPolicy
        metadata:
          name: operator
          # namespace: # will be supplied by the controller via the namespaceSelector
        spec:
          remediationAction: "inform"
          object-templates:
          - complianceType: "musthave" # at this level, it means the role must exist and must
have the following rules
            apiVersion: rbac.authorization.k8s.io/v1
            kind: Role
            metadata:
              name: example
            objectDefinition:
              rules:
                - complianceType: "musthave" # at this level, it means if the role exists the rule is a
musthave
                  apiGroups: ["extensions", "apps"]
                  resources: ["deployments"]
                  verbs: ["get", "list", "watch", "create", "delete","patch"]
```

3. Define a **PlacementBinding** resource to bind your policy to your **Placement** resource. Your **PlacementBinding** resource might resemble the following YAML sample:

```
apiVersion: policy.open-cluster-management.io/v1
kind: PlacementBinding
metadata:
  name: binding1
placementRef:
  name: placement1
  apiGroup: cluster.open-cluster-management.io
  kind: Placement
subjects:
- name: policy1
  apiGroup: policy.open-cluster-management.io
  kind: Policy
```

4.15.1.1.1. Viewing your security policy from the CLI

Complete the following steps to view your security policy from the CLI:

1. View details for a specific security policy by running the following command:

```
oc get policies.policy.open-cluster-management.io <policy-name> -n <policy-namespace> -o
yaml
```

2. View a description of your security policy by running the following command:

```
oc describe policies.policy.open-cluster-management.io <policy-name> -n <policy-
namespace>
```

4.15.1.2. Creating a cluster security policy from the console

After you log in to your Red Hat Advanced Cluster Management, navigate to the *Governance* page and click **Create policy**. As you create your new policy from the console, a YAML file is also created in the YAML editor. To view the YAML editor, select the toggle at the beginning of the *Create policy* form to enable it.

1. Complete the *Create policy* form, then select the **Submit** button. Your YAML file might resemble the following policy:

```
apiVersion: policy.open-cluster-management.io/v1
kind: Policy
metadata:
  name: policy-pod
  annotations:
    policy.open-cluster-management.io/categories:
'SystemAndCommunicationsProtections,SystemAndInformationIntegrity'
    policy.open-cluster-management.io/controls: 'control example'
    policy.open-cluster-management.io/standards: 'NIST,HIPAA'
    policy.open-cluster-management.io/description:
spec:
  complianceType: musthave
  namespaces:
    exclude: ["kube*"]
```

```
    include: ["default"]
    pruneObjectBehavior: None
  object-templates:
  - complianceType: musthave
    objectDefinition:
      apiVersion: v1
      kind: Pod
      metadata:
        name: pod1
      spec:
        containers:
        - name: pod-name
          image: 'pod-image'
          ports:
          - containerPort: 80
  remediationAction: enforce
  disabled: false
```

See the following **PlacementBinding** example:

```
apiVersion: policy.open-cluster-management.io/v1
kind: PlacementBinding
metadata:
  name: binding-pod
placementRef:
  name: placement-pod
  kind: Placement
  apiGroup: cluster.open-cluster-management.io
subjects:
- name: policy-pod
  kind: Policy
  apiGroup: policy.open-cluster-management.io
```

See the following **Placement** example:

```
apiVersion: cluster.open-cluster-management.io/v1beta1
 kind: Placement
 metadata:
   name: placement-pod
spec:
  predicates:
  - requiredClusterSelector:
      labelSelector:
        matchLabels:
          cloud: "IBM"
      tolerations:
        - key: cluster.open-cluster-management.io/unavailable
          operator: Exists
        - key: cluster.open-cluster-management.io/unreachable
          operator: Exists
```

2. **Optional:** Add a description for your policy.

3. Click **Create Policy**. A security policy is created from the console.

4.15.1.2.1. Viewing your security policy from the console

View any security policy and the status from the console.

1. Navigate to the *Governance* page to view a table list of your policies. **Note:** You can filter the table list of your policies by selecting the *Policies* tab or *Cluster violations* tab.

2. Select one of your policies to view more details. The *Details*, *Clusters*, and *Templates* tabs are displayed. When the cluster or policy status cannot be determined, the following message is displayed: **No status**.

3. Alternatively, select the *Policies* tab to view the list of policies. Expand a policy row to view the *Description*, *Standards*, *Controls*, and *Categories* details.

4.15.1.3. Creating policy sets from the CLI

By default, the policy set is created with no policies or placements. You must create a placement for the policy set and have at least one policy that exists on your cluster. When you create a policy set, you can add numerous policies.

Run the following command to create a policy set from the CLI:

```
oc apply -f <policyset-filename>
```

4.15.1.4. Creating policy sets from the console

1. From the navigation menu, select **Governance**.

2. Select the *Policy sets* tab.

3. Select the **Create policy set**button and complete the form.

4. Add the details for your policy set and select the **Submit** button.

Your policy is listed from the policy table.

4.15.2. Updating security policies

Learn to update security policies.

4.15.2.1. Adding a policy to a policy set from the CLI

1. Run the following command to edit your policy set:

   ```
   oc edit policysets <your-policyset-name>
   ```

2. Add the policy name to the list in the **policies** section of the policy set.

3. Apply your added policy in the placement section of your policy set with the following command:

```
oc apply -f <your-added-policy.yaml>
```

PlacementBinding and **Placement** are both created.

Note: If you delete the placement binding, the policy is still placed by the policy set.

4.15.2.2. Adding a policy to a policy set from the console

1. Add a policy to the policy set by selecting the *Policy sets* tab.
2. Select the Actions icon and select **Edit**. The *Edit policy set* form appears.
3. Navigate to the *Policies* section of the form to select a policy to add to the policy set.

4.15.2.3. Disabling security policies

Your policy is enabled by default. Disable your policy from the console.

After you log in to your Red Hat Advanced Cluster Management for Kubernetes console, navigate to the *Governance* page to view a table list of your policies.

Select the **Actions** icon > **Disable policy**. The *Disable Policy* dialog box appears.

Click **Disable policy**. Your policy is disabled.

4.15.3. Deleting a security policy

Delete a security policy from the CLI or the console.

Use the following procedure to delete from the CLI:

1. Verify that your policy is removed by running the following command: **oc get policies.policy.open-cluster-management.io <policy-name> -n <policy-namespace>**

Use the following procedure to delete a security policy from the console.

1. From the navigation menu, click **Governance** to view a table list of your policies.
2. Click the **Actions** icon for the policy you want to delete in the policy violation table.
3. Click **Remove**.
4. From the *Remove policy* dialog box, click **Remove policy**.

4.15.3.1. Deleting policy sets from the console

1. From the *Policy sets* tab, select the **Actions** icon for the policy set. When you click **Delete**, the *Permanently delete Policyset?* dialogue box appears.
2. Click the **Delete** button.

4.15.4. Cleaning up resources that are created by policies

Use the **pruneObjectBehavior** parameter in a configuration policy to clean up resources that are created by the policy. When **pruneObjectBehavior** is set, the related objects are only cleaned up after the configuration policy (or parent policy) associated with them is deleted.

View the following descriptions of the values that can be used for the parameter:

- **DeleteIfCreated**: Cleans up any resources created by the policy.
- **DeleteAll**: Cleans up all resources managed by the policy.
- **None**: This is the default value and maintains the same behavior from previous releases, where no related resources are deleted.

You can set the value directly in the YAML file as you create a policy from the command line.

From the console, you can select the value in the *Prune Object Behavior* section of the *Policy templates* step.

Notes:

- If a policy that installs an operator has the **pruneObjectBehavior** parameter defined, then additional clean up is needed to complete the operator uninstall. You might need to delete the operator **ClusterServiceVersion** object as part of this cleanup.
- As you disable the **config-policy-addon** resource on the managed cluster, the **pruneObjbectBehavior** is ignored. To automatically clean up the related resources on the policies, you must remove the policies from the managed cluster before the add-on is disabled.

4.15.5. Policy command line interface

With the **policytools** command line interface (CLI), you can interact with policies locally to help with creating and debugging.

template-resolver

The **template-resolver** is a subcommand for **policytools** that resolves managed cluster and hub cluster templates that are embedded in policies. The **template-resolver** reads from either a file, or from standard input.
To resolve a policy with hub cluster templates, you must provide the **--cluster-name** argument with the name of a managed cluster that is imported into Red Hat Advanced Cluster Management, and you must provide the **--hub-kubeconfig** argument with the path to a **kubeconfig** file that references the hub cluster.

The **policytools** CLI is available for download from the hub cluster console. See Command line tools.

4.15.6. Additional resources

- View more descriptions of the policy YAML files in the Hub cluster policy framework [Policy overview].
- See Resources that support support set-based requirements in the Kubernetes documentation for a valid expression.
- View the stable **Policysets**, which require the Policy Generator for deployment, PolicySets--Stable.
- Refer to Governance for more topics about policies.

4.15.7. Managing operator policies in disconnected environments

You might need to deploy Red Hat Advanced Cluster Management for Kubernetes policies on Red Hat OpenShift Container Platform clusters that are not connected to the internet (disconnected). If the policies you deploy are used to deploy policies that install an Operator Lifecycle Manager operator, you must follow the procedure for Mirroring an Operator catalog .

Complete the following steps to validate access to the operator images:

1. See Verify required packages are available to validate that packages you require to use with policies are available. You must validate availability for each image registry used by any managed cluster that the following policies are deployed to:
   - **container-security-operator**
   - Deprecated: **gatekeeper-operator-product**
   - **compliance-operator**
2. See Configure image content source policies to validate that the sources are available. The image content source policies must exist on each of the disconnected managed clusters and can be deployed using a policy to simplify the process. See the following table of image source locations:

Governance policy type	Image source location
Container security	**registry.redhat.io/quay**
Compliance	**registry.redhat.io/compliance**
Gatekeeper	**registry.redhat.io/rhacm2**

4.15.8. Installing Red Hat OpenShift Platform Plus by using a policy set

Continue reading for guidance to apply the Red Hat Openshift Platform Plus policy set. When you apply the Red Hat OpenShift policy set, the Red Hat Advanced Cluster Security secured cluster services and the Compliance Operator are deployed onto all of your OpenShift Container Platform managed clusters.

4.15.8.1. Prerequisites

Complete the following steps before you apply the policy set:

1. To allow for subscriptions to be applied to your cluster, you must apply the **policy-configure-subscription-admin-hub.yaml** policy and set the remediation action to **enforce**. Copy and paste the following YAML into the YAML editor of the console:

```
apiVersion: policy.open-cluster-management.io/v1
kind: Policy
metadata:
  name: policy-configure-subscription-admin-hub
  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
spec:
  remediationAction: inform
```

```
  disabled: false
  policy-templates:
    - objectDefinition:
        apiVersion: policy.open-cluster-management.io/v1
        kind: ConfigurationPolicy
        metadata:
          name: policy-configure-subscription-admin-hub
        spec:
          remediationAction: inform
          severity: low
          object-templates:
            - complianceType: musthave
              objectDefinition:
                apiVersion: rbac.authorization.k8s.io/v1
                kind: ClusterRole
                metadata:
                  name: open-cluster-management:subscription-admin
                rules:
                - apiGroups:
                  - app.k8s.io
                  resources:
                  - applications
                  verbs:
                  - '*'
                - apiGroups:
                  - apps.open-cluster-management.io
                  resources:
                  - '*'
                  verbs:
                  - '*'
                - apiGroups:
                  - ""
                  resources:
                  - configmaps
                  - secrets
                  - namespaces
                  verbs:
                  - '*'
            - complianceType: musthave
              objectDefinition:
                apiVersion: rbac.authorization.k8s.io/v1
                kind: ClusterRoleBinding
                metadata:
                  name: open-cluster-management:subscription-admin
                roleRef:
                  apiGroup: rbac.authorization.k8s.io
                  kind: ClusterRole
                  name: open-cluster-management:subscription-admin
                subjects:
                - apiGroup: rbac.authorization.k8s.io
                  kind: User
                  name: kube:admin
                - apiGroup: rbac.authorization.k8s.io
                  kind: User
                  name: system:admin
---
```

```
apiVersion: policy.open-cluster-management.io/v1
kind: PlacementBinding
metadata:
  name: binding-policy-configure-subscription-admin-hub
placementRef:
  name: placement-policy-configure-subscription-admin-hub
  kind: Placement
  apiGroup: cluster.open-cluster-management.io
subjects:
- name: policy-configure-subscription-admin-hub
  kind: Policy
  apiGroup: policy.open-cluster-management.io
---
apiVersion: cluster.open-cluster-management.io/v1beta1
kind: Placement
metadata:
  name: placement-policy-configure-subscription-admin-hub
spec:
  predicates:
  - requiredClusterSelector:
      labelSelector:
        matchExpressions:
        - {key: name, operator: In, values: ["local-cluster"]}
    tolerations:
      - key: cluster.open-cluster-management.io/unavailable
        operator: Exists
      - key: cluster.open-cluster-management.io/unreachable
        operator: Exists
```

2. To apply the previous YAML from the command line interface, run the following command:

```
oc apply -f policy-configure-subscription-admin-hub.yaml
```

3. Install the Policy Generator kustomize plugin. Use Kustomize v4.5 or newer. See Generating a policy to install an Operator.

4. Policies are installed to the **policies** namespace. You must bind that namespace to a **ClusterSet**. For example, copy and apply the following example YAML to bind the namespace to the default **ClusterSet**:

```
apiVersion: cluster.open-cluster-management.io/v1beta2
kind: ManagedClusterSetBinding
metadata:
   name: default
   namespace: policies
spec:
   clusterSet: default
```

5. Run the following command to apply the **ManagedClusterSetBinding** resource from the command line interface:

```
oc apply -f managed-cluster.yaml
```

After you meet the prerequisite requirements, you can apply the policy set.

4.15.8.2. Applying Red Hat OpenShift Platform Plus policy set

1. Use the **openshift-plus/policyGenerator.yaml** file that includes the prerequisite configuration for Red Hat OpenShift Plus. See **openshift-plus/policyGenerator.yaml**.

2. Apply the policies to your hub cluster by using the **kustomize** command:

```
kustomize build --enable-alpha-plugins  | oc apply -f -
```

Note: For any components of OpenShift Platform Plus that you do not want to install, edit the **policyGenerator.yaml** file and remove or comment out the policies for those components.

4.15.8.3. Additional resources

- See Red Hat OpenShift Platform Plus policy set for an overview of the policy set.
- Return to the beginning of the topic, Installing Red Hat OpenShift Platform Plus by using a policy set

4.15.9. Installing an operator by using the *OperatorPolicy* resource

To install Operator Lifecycle Manager (OLM) managed operators on your managed clusters, use an **OperatorPolicy** policy template in a **Policy** definition.

4.15.9.1. Creating an *OperatorPolicy* resource to install Quay

See the following operator policy sample that installs the latest Quay operator in the **stable-3.11** channel using the Red Hat operator catalog:

```
apiVersion: policy.open-cluster-management.io/v1
kind: Policy
metadata:
  name: install-quay
  namespace: open-cluster-management-global-set
spec:
  disabled: false
  policy-templates:
    - objectDefinition:
        apiVersion: policy.open-cluster-management.io/v1beta1
        kind: OperatorPolicy
        metadata:
          name: install-quay
        spec:
          remediationAction: enforce
          severity: critical
          complianceType: musthave
          upgradeApproval: None
          subscription:
            channel: stable-3.11
            name: quay-operator
            source: redhat-operators
            sourceNamespace: openshift-marketplace
```

After you add the **OperatorPolicy** policy template, the **operatorGroup** and **subscription** objects are

created on the cluster by using the controller. As a result, the rest of the installation is completed by OLM. You can view the health of owned resources in the **.status.Conditions** and **.status.relatedObjects** fields of the **OperatorPolicy** resource on your managed cluster.

To verify the operator policy status, run the following command on your managed cluster:

```
oc -n <managed cluster namespace> get operatorpolicy install-quay
```

4.15.9.2. Additional resources

See Operator policy controller

4.16. SECURING THE HUB CLUSTER

Secure your Red Hat Advanced Cluster Management for Kubernetes installation by enhancing the hub cluster security. Complete the following steps:

1. Secure Red Hat OpenShift Container Platform. For more information, see Security and compliance in the OpenShift Container Platform documentation.
2. Setup role-based access control (RBAC). For more information, see Role-based access control.
3. Customize certificates, see Certificates.
4. Define your cluster credentials, see Managing credentials overview
5. Review the policies that are available to help you harden your cluster security. See Supported policies.

CHAPTER 5. GATEKEEPER OPERATOR OVERVIEW

The Gatekeeper operator installs Gatekeeper, which is a validating webhook with auditing capabilities. Install the Gatekeeper operator on a Red Hat OpenShift Container Platform cluster from the Operator Lifecycle Manager operator catalog. With Red Hat Advanced Cluster Management for Kubernetes, you can install Gatekeeper on your hub cluster by using the Gatekeeper operator policy. After you install Gatekeeper, use it for the following benefits:

- Deploy and check Gatekeeper **ConstraintTemplates** and constraints on managed clusters by using the Red Hat Advanced Cluster Management policy integration.
- Enforce Kubernetes custom resource definition-based policies that run with your Open Policy Agent (OPA).
- Evaluate Kubernetes resource compliance requests for the Kubernetes API by using the Gatekeeper constraints.
- Use OPA as the policy engine and use Rego as the policy language.

Prerequisite: You need a Red Hat Advanced Cluster Management for Kubernetes or Red Hat OpenShift Container Platform Plus subscription to install Gatekeeper and apply Gatekeeper policies to your cluster.

To learn more about using the Gatekeeper operator, see the following resources:

- General support
- Operator channels
- Configuring the Gatekeeper operator
- Managing the Gatekeeper operator installation policies
- Integrating Gatekeeper constraints and constraint templates

5.1. GENERAL SUPPORT

To understand the support you receive from the Gatekeeper operator, see the following list:

- Supports current version of the Gatekeeper operator, preceding versions, and all z-stream releases of those versions.
- Receive maintenance support and relevant security vulnerability fixes for preceding and current versions.
- Support for all Red Hat OpenShift Container Platform versions that receive standard support. **Note**: The Gatekeeper operator is not supported on end-of-life OpenShift Container Platform versions or versions that receive extended support.

To view the release notes for the Gatekeeper operator, see gatekeeper-operator-bundle.

5.2. OPERATOR CHANNELS

With the Gatekeeper operator, you have access to two types of channels to help you make upgrades. These channels are the **stable** channel and the **y-stream version** channel.

With the **stable** channel, you can access the latest available version, whether it is an **x-stream**, **y-stream**, or **z-stream**. The **stable** channel includes the latest version of the latest **y-stream** channel.

With the **y-stream version** channel, you can access all the **z-stream** versions for a particular **y-stream**.

5.3. CONFIGURING THE GATEKEEPER OPERATOR

Install the Gatekeeper operator from the Operator Lifecycle Manager catalog to install Gatekeeper on your cluster. With Red Hat Advanced Cluster Management you can use a policy to install the Gatekeeper operator by using the governance framework. After you install the Gatekeeper operator, configure the Gatekeeper operator custom resource to install Gatekeeper.

5.3.1. Prerequisites

- **Required access**: Cluster administrator.
- Understand how to use the Operator Lifecycle Manager and the OperatorHub by completing the *Adding Operators to a cluster* and the *Additional resources* section in the OpenShift Container Platform documentation.

5.3.2. Gatekeeper custom resource sample

The Gatekeeper operator custom resource tells the Gatekeeper operator to start the Gatekeeper installation on the cluster. To install Gatekeeper, use the following sample YAML, which includes sample and default values:

```
apiVersion: operator.gatekeeper.sh/v1alpha1
kind: Gatekeeper
metadata:
  name: gatekeeper
spec:
  audit:
    auditChunkSize: 66
    auditEventsInvolvedNamespace: Enabled (1)
    auditFromCache: Enabled
    auditInterval: 10s
    constraintViolationLimit: 55
    containerArguments: (2)
    - name: ""
      value: ""
    resources:
      limits:
        cpu: 500m
        memory: 150Mi
      requests:
        cpu: 500m
        memory: 130Mi
    emitAuditEvents: Enabled
    logLevel: DEBUG
    podAnnotations: {} (3)
    replicas: 1
  validatingWebhook: Enabled
  mutatingWebhook: Enabled
  webhook:
```

```
    admissionEventsInvolvedNamespace: Enabled 4
    containerArguments: 5
    - name: ""
      value: ""
    disabledBuiltins:
     - http.send
    emitAdmissionEvents: Enabled
    failurePolicy: Fail
    operations: 6
     - CREATE
     - UPDATE
     - CONNECT
    replicas: 3
    resources:
      limits:
        cpu: 480m
        memory: 140Mi
      requests:
        cpu: 400m
        memory: 120Mi
    rules: 7
    - operations: []
      resources: []
    timeoutSeconds: 3 8
  mutatingWebhookConfig: 9
    failurePolicy: ""
    logMutations: Disabled
    mutationAnnotations: Disabled
    namespaceSelector: {}
    operations: []
    rules: 10
    - operations: []
      resources: []
    timeoutSeconds: 1 11
  config: 12
    matches:
      - excludedNamespaces: ["test-*", "my-namespace"]
        processes: ["*"]
    disableDefaultMatches: false 13
  nodeSelector:
    region: "EMEA"
  affinity:
    podAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        - labelSelector:
            matchLabels:
              auditKey: "auditValue"
          topologyKey: topology.kubernetes.io/zone
  tolerations:
    - key: "Example"
      operator: "Exists"
      effect: "NoSchedule"
  podAnnotations:
    some-annotation: "this is a test"
    other-annotation: "another test"
```

```
-
```

1. Enable the **auditEventsInvolvedNamespace** parameter to manage the namespace audit event you want to create. When you enable this parameter, the Gatekeeper controller deployment runs with the following argument: **--audit-events-involved-namespace=true**.

3. For version 3.20 and later, specify **audit.podAnnotations** specific to the audit pod by providing a list of annotation names and values to add to the pod. An omitted value is treated as **true**.

4. Enable the **admissionEventsInvolvedNamespace** parameter to manage the namespace admission event you want to create. When you enable this parameter, the Gatekeeper controller deployment runs with the following argument: **--admission-events-involved-namespace=true**.

2 5. Specify **containerArguments** by providing a list of argument names and values to pass to the container. Omit leading dashes from the argument name. An omitted value is treated as **true**. Arguments that you provide are ignored if the argument is set previously by the operator or configurations from other fields. See the following list of flags that are deny-listed and are not currently supported:

    - **port**
    - **prometheus-port**
    - **health-addr**
    - **validating-webhook-configuration-name**
    - **mutating-webhook-configuration-name**
    - **disable-cert-rotation**
    - **client-cert-name**
    - **tls-min-version**

6. Use **operations** to manage the operations for which the webhook is invoked. For example, **CREATE**, **UPDATE**, **CONNECT**, and **DELETE**.

7 10. For version 3.20 and later, specify **rules** to overwrite the rules configuration in the webhook configuration. When set, this field takes precedence over the Operations field.

8 11. For version 3.20 and later, specify the timeout for the webhook configuration.

9. For version 3.20 and later, use the **mutatingWebhookConfig** section to configure the mutating webhook with configurations that override the values from **spec.webhook** or are specific to the mutating webhook.

12. Use the **config** section to exclude namespaces from certain processes for all constraints on your cluster.

13. The **disableDefaultMatches** parameter is a boolean parameter that disables appending the default exempt namespaces provided by the Gatekeeper operator. The default exempt namespaces are OpenShift Container Platform or Kubernetes system namespaces. By default, this parameter is set to **false** to allow the default namespaces to be appended.

5.3.3. Configuring *auditFromCache* for sync details

The Gatekeeper operator exposes a setting in the Gatekeeper operator custom resource for the audit configuration with the **auditFromCache** parameter, which is disabled by default. Configure the **auditFromCache** parameter to collect resources from constraints.

When you set the **auditFromCache** parameter to **Automatic**, the Gatekeeper operator collects resources from constraints and inserts those resources into your Gatekeeper **Config** resource. If the resource does not exist, the Gatekeeper operator creates the **Config** resource.

If you set the **auditFromCache** parameter to **Enabled**, you need to manually set the Gatekeeper **Config** resource with the objects to sync to the cache. For more information, see *Configuring Audit* in the Gatekeeper documentation.

To configure the **auditFromCache** parameter for resource collection from constraints, complete the following steps:

1. Set **auditFromCache** to **Automatic** in the **Gatekeeper** resource. See the following example:

```
apiVersion: operator.gatekeeper.sh/v1alpha1
kind: Gatekeeper
metadata:
  name: gatekeeper
spec:
  audit:
    replicas: 2
    logLevel: DEBUG
    auditFromCache: Automatic
```

2. To verify that the resources are added to your **Config** resource, view that the **syncOnly** parameter section is added. Run the following command:

```
oc get configs.config.gatekeeper.sh config -n openshift-gatekeeper-system
```

   Your **Config** resource might resemble the following example:

```
apiVersion: config.gatekeeper.sh/v1alpha1
kind: Config
metadata:
 name: config
 namespace: "openshift-gatekeeper-system"
spec:
 sync:
   syncOnly:
   - group: ""
     version: "v1"
     kind: "Namespace"
   - group: ""
     version: "v1"
     kind: "Pod"
```

Optional: You can view the explanation of the **auditFromCache** setting from the description of the Gatekeeper operator custom resource by running the following command:

```
oc explain gatekeeper.spec.audit.auditFromCache
```

5.3.4. Additional resources

- For more information, see Configuring Audit in the Gatekeeper documentation.

5.4. MANAGING THE GATEKEEPER OPERATOR INSTALLATION POLICIES

Use the Red Hat Advanced Cluster Management policy to install the Gatekeeper operator and Gatekeeper on a managed cluster.

Required access: Cluster administrator

To create, view, and update your Gatekeeper operator installation policies, complete the following sections:

- Installing Gatekeeper using a Gatekeeper operator policy
- Creating a Gatekeeper policy from the console
- Upgrading Gatekeeper and the Gatekeeper operator
- Disabling Gatekeeper operator policy
- Deleting Gatekeeper operator policy
- Uninstalling Gatekeeper constraints, Gatekeeper instance, and Gatekeeper operator policy

5.4.1. Installing Gatekeeper using a Gatekeeper operator policy

To install the Gatekeeper operator policy, use the configuration policy controller. During the install, the operator group and subscription pull the Gatekeeper operator to install it on your managed cluster. Then, the policy creates a Gatekeeper custom resource to configure Gatekeeper.

The Red Hat Advanced Cluster Management configuration policy controller checks the Gatekeeper operator policy and supports the **enforce** remediation action. When you set the controller to **enforce** it automatically creates the Gatekeeper operator objects on the managed cluster.

5.4.2. Creating a Gatekeeper policy from the console

When you create a Gatekeeper policy from the console, you must set your remediation **enforce** to install Gatekeeper.

5.4.2.1. Viewing the Gatekeeper operator policy

To view your Gatekeeper operator policy and its status from the console, complete the following steps:

1. Select the **policy-gatekeeper-operator** policy to view more details.
2. Select the *Clusters* tab to view the policy violations.

5.4.3. Upgrading Gatekeeper and the Gatekeeper operator

You can upgrade the versions for Gatekeeper and the Gatekeeper operator. When you install the Gatekeeper operator with the Gatekeeper operator policy, notice the value for **upgradeApproval**. The operator upgrades automatically when you set **upgradeApproval** to **Automatic**.

If you set **upgradeApproval** to **Manual**, you must manually approve the upgrade for each cluster where the Gatekeeper operator is installed.

5.4.4. Disabling Gatekeeper operator policy

To disable your **policy-gatekeeper-operator** policy, select the **Disable** option from the *Actions* menu in the console, or set **spec.disabled: true** from the CLI.

5.4.5. Deleting Gatekeeper operator policy

To delete your Gatekeeper operator policy from your CLI, complete the following steps:

1. Delete Gatekeeper operator policy by running the following command:

```
oc delete policies.policy.open-cluster-management.io <policy-gatekeeper-operator-name> -n <namespace>
```

2. Verify that you deleted your policy by running the following command:

```
oc get policies.policy.open-cluster-management.io <policy-gatekeeper-operator-name> -n <namespace>
```

To delete your Gatekeeper operator policy from the console, click the **Actions** icon for the **policy-gatekeeper-operator** policy and select **Delete**.

5.4.6. Uninstalling Gatekeeper constraints, Gatekeeper instance, and Gatekeeper operator policy

To uninstall Gatekeeper policy, complete the steps in the following sections:

- Removing Gatekeeper constraints
- Removing Gatekeeper instance
- Removing Gatekeeper operator

5.4.6.1. Removing Gatekeeper constraints

To remove the Gatekeeper constraint and **ConstraintTemplate** from your managed cluster, complete the following steps:

1. Edit your Gatekeeper constraint or **ConstraintTemplate** policy.
2. Locate the template that you used to create the Gatekeeper **Constraint** and **ConstraintTemplate**.
3. Delete the entries from the list of templates. (Or delete the policy if they're the only templates.)
4. Save and apply the policy.

Note: The constraint and **ConstraintTemplate** are provided directly in the **policy-templates** instead of within a **ConfigurationPolicy**.

5.4.6.2. Removing Gatekeeper instance

To remove the Gatekeeper instance from your managed cluster, complete the following steps:

1. Edit your Gatekeeper operator policy.
2. Locate the **ConfigurationPolicy** template that you used to create the Gatekeeper operator custom resource.
3. Change the value for **complianceType** of the **ConfigurationPolicy** template to **mustnothave**. Changing the value deletes the Gatekeeper operator custom resource, signaling to the Gatekeeper operator to clean up the Gatekeeper deployment.

5.4.6.3. Removing Gatekeeper operator

To remove the Gatekeeper operator from your managed cluster, complete the following steps:

1. Edit your Gatekeeper operator policy.
2. Locate the **OperatorPolicy** template that you used to create the Subscription CR.
3. Change the value for **complianceType** of the **OperatorPolicy** template to **mustnothave**.

5.4.7. Additional resources

For more details, see the following resources:

- Integrating Gatekeeper constraints and constraint templates.
- Policy Gatekeeper.
- For an explanation of the optional parameters that can be used for the Gatekeeper operator policy, see Gatekeeper Helm Chart.

5.5. INTEGRATING GATEKEEPER CONSTRAINTS AND CONSTRAINT TEMPLATES

To create Gatekeeper policies, use **ConstraintTemplates** and constraints. Add templates and constraints to the **policy-templates** of a **Policy** resource. View the following YAML examples that use Gatekeeper constraints in Red Hat Advanced Cluster Management policies:

- **ConstraintTemplates** and constraints: Use the Gatekeeper integration feature by using Red Hat Advanced Cluster Management policies for multicluster distribution of Gatekeeper constraints and Gatekeeper audit results aggregation on the hub cluster. The following example defines a Gatekeeper **ConstraintTemplate** and constraint (**K8sRequiredLabels**) to ensure the **gatekeeper** label is set on all namespaces:

```
apiVersion: policy.open-cluster-management.io/v1
kind: Policy
metadata:
  name: require-gatekeeper-labels-on-ns
```

```
spec:
  remediationAction: inform (1)
  disabled: false
  policy-templates:
    - objectDefinition:
        apiVersion: templates.gatekeeper.sh/v1beta1
        kind: ConstraintTemplate
        metadata:
          name: k8srequiredlabels
          annotations:
            policy.open-cluster-management.io/severity: low (2)
        spec:
          crd:
            spec:
              names:
                kind: K8sRequiredLabels
              validation:
                openAPIV3Schema:
                  properties:
                    labels:
                      type: array
                      items: string
          targets:
            - target: admission.k8s.gatekeeper.sh
              rego: |
                package k8srequiredlabels
                violation[{"msg": msg, "details": {"missing_labels": missing}}] {
                  provided := {label | input.review.object.metadata.labels[label]}
                  required := {label | label := input.parameters.labels[_]}
                  missing := required - provided
                  count(missing) > 0
                  msg := sprintf("you must provide labels: %v", [missing])
                }
    - objectDefinition:
        apiVersion: constraints.gatekeeper.sh/v1beta1
        kind: K8sRequiredLabels
        metadata:
          name: ns-must-have-gk
          annotations:
            policy.open-cluster-management.io/severity: low (3)
        spec:
          enforcementAction: dryrun
          match:
            kinds:
              - apiGroups: [""]
                kinds: ["Namespace"]
          parameters:
            labels: ["gatekeeper"]
```

(1) Since the **remediationAction** is set to **inform**, the **enforcementAction** field of the Gatekeeper constraint is overridden to **warn**. This means that Gatekeeper detects and warns you about creating or updating a namespace that is missing the **gatekeeper** label. If the policy **remediationAction** is set to **enforce**, the Gatekeeper constraint **enforcementAction** field is overridden to **deny**. In this context, this configuration prevents any user from creating or updating a namespace that is missing the **gatekeeper** label.

2 3 **Optional:** Set a severity value for the **policy.open-cluster-management.io/severity** annotation for each Gatekeeper constraint or constraint template. Valid values are the

With the previous policy, you might receive the following policy status message: **warn - you must provide labels: {"gatekeeper"} (on Namespace default); warn - you must provide labels: {"gatekeeper"} (on Namespace gatekeeper-system)**. When you delete Gatekeeper constraints or **ConstraintTemplates** from a policy, the constraints and **ConstraintTemplates** are also deleted from your managed cluster.

To view the Gatekeeper audit results for a specific managed cluster from the console, go to to the policy template *Results* page. If search is enabled, view the YAML of the Kubernetes objects that failed the audit.

Notes:

- The *Related resources* section is only available when Gatekeeper generates audit results.
- The Gatekeeper audit runs every minute by default. Audit results are sent back to the hub cluster to be viewed in the Red Hat Advanced Cluster Management policy status of the managed cluster.

- **policy-gatekeeper-admission**: Use the **policy-gatekeeper-admission** configuration policy within a Red Hat Advanced Cluster Management policy to check for Kubernetes API requests denied by the Gatekeeper admission webhook. View the following example:

```
apiVersion: policy.open-cluster-management.io/v1
kind: ConfigurationPolicy
metadata:
  name: policy-gatekeeper-admission
spec:
  remediationAction: inform 1
  severity: low
  object-templates:
    - complianceType: mustnothave
      objectDefinition:
        apiVersion: v1
        kind: Event
        metadata:
          namespace: openshift-gatekeeper-system 2
          annotations:
            constraint_action: deny
            constraint_kind: K8sRequiredLabels
            constraint_name: ns-must-have-gk
            event_type: violation
```

1 The **ConfigurationPolicy remediationAction** parameter is overwritten by **remediationAction** in the parent policy.

2 Set to the actual namespace where Gatekeeper is running if it is different.

5.5.1. Additional resources

For more details, see the following resources:

- **policy-gatekeeper-operator.yaml**
- What is OPA Gatekeeper?
- Creating configuration policies
- Governance

CHAPTER 6. SECURITY OVERVIEW

Manage the security of your Red Hat Advanced Cluster Management for Kubernetes components. Govern your cluster with defined policies and processes to identify and minimize risks. Use policies to define rules and set controls.

Prerequisite: You must configure authentication service requirements for Red Hat Advanced Cluster Management for Kubernetes. See Access control for more information.

Read through the following topics to learn more about securing your cluster:

- Certificates introduction
- Governance

6.1. CERTIFICATES INTRODUCTION

You can use various certificates to verify authenticity for your Red Hat Advanced Cluster Management for Kubernetes cluster. Continue reading to learn about certificate management.

- Certificates
- Managing certificates

6.1.1. Certificates

All certificates required by services that run on Red Hat Advanced Cluster Management are created during the installation of Red Hat Advanced Cluster Management. View the following list of certificates, which are created and managed by the following components of Red Hat OpenShift Container Platform:

- OpenShift Service Serving Certificates
- Red Hat Advanced Cluster Management webhook controllers
- Kubernetes Certificates API
- OpenShift default ingress

Required access: Cluster administrator

Continue reading to learn more about certificate management:

- Red Hat Advanced Cluster Management hub cluster certificates
- Red Hat Advanced Cluster Management managed certificates

Note: Users are responsible for certificate rotations and updates.

6.1.1.1. Red Hat Advanced Cluster Management hub cluster certificates

OpenShift default ingress certificate is technically a hub cluster certificate. After the Red Hat Advanced Cluster Management installation, observability certificates are created and used by the observability components to provide mutual TLS on the traffic between the hub cluster and managed cluster.

- The **open-cluster-management-observability** namespace contains the following certificates:

    - **observability-server-ca-certs**: Has the CA certificate to sign server-side certificates
    - **observability-client-ca-certs**: Has the CA certificate to sign client-side certificates
    - **observability-server-certs**: Has the server certificate used by the **observability-observatorium-api** deployment
    - **observability-grafana-certs**: Has the client certificate used by the **observability-rbac-query-proxy** deployment

- The **open-cluster-management-addon-observability** namespace contain the following certificates on managed clusters:
    - **observability-managed-cluster-certs**: Has the same server CA certificate as **observability-server-ca-certs** in the hub server
    - **observability-controller-open-cluster-management.io-observability-signer-client-cert**: Has the client certificate used by the **metrics-collector-deployment**

The CA certificates are valid for five years and other certificates are valid for one year. All observability certificates are automatically refreshed upon expiration. View the following list to understand the effects when certificates are automatically renewed:

- Non-CA certificates are renewed automatically when the remaining valid time is no more than 73 days. After the certificate is renewed, the pods in the related deployments restart automatically to use the renewed certificates.
- CA certificates are renewed automatically when the remaining valid time is no more than one year. After the certificate is renewed, the old CA is not deleted but co-exist with the renewed ones. Both old and renewed certificates are used by related deployments, and continue to work. The old CA certificates are deleted when they expire.
- When a certificate is renewed, the traffic between the hub cluster and managed cluster is not interrupted.

View the following Red Hat Advanced Cluster Management hub cluster certificates table:

Table 6.1. Red Hat Advanced Cluster Management hub cluster certificates

Namespace	Secret name	Pod label	
open-cluster-management	channels-apps-open-cluster-management-webhook-svc-ca	app=multicluster-operators-channel	open-cluster-management
channels-apps-open-cluster-management-webhook-svc-signed-ca	app=multicluster-operators-channel	open-cluster-management	multicluster-operators-application-svc-ca
app=multicluster-operators-application	open-cluster-management	multicluster-operators-application-svc-signed-ca	app=multicluster-operators-application

Namespace	Secret name	Pod label	
open-cluster-management-hub	registration-webhook-serving-cert signer-secret	Not required	open-cluster-management-hub

6.1.1.2. Red Hat Advanced Cluster Management managed certificates

View the following table for a summarized list of the component pods that contain Red Hat Advanced Cluster Management managed certificates and the related secrets:

Table 6.2. Pods that contain Red Hat Advanced Cluster Management managed certificates

Namespace	Secret name (if applicable)
open-cluster-management-agent-addon	cluster-proxy-open-cluster-management.io-proxy-agent-signer-client-cert
open-cluster-management-agent-addon	cluster-proxy-service-proxy-server-certificates

6.1.1.2.1. Managed cluster certificates

You can use certificates to authenticate managed clusters with the hub cluster. Therefore, it is important to be aware of troubleshooting scenarios associated with these certificates.

The managed cluster certificates are refreshed automatically.

6.1.1.3. Additional resources

- Use the certificate policy controller to create and manage certificate policies on managed clusters. See Certificate policy controller for more details.
- See Using custom CA certificates for a secure HTTPS connection for more details about securely connecting to a privately-hosted Git server with SSL/TLS certificates.
- See OpenShift Service Serving Certificates for more details.
- The OpenShift Container Platform default ingress is a hub cluster certificate. See Replacing the default ingress certificate for more details.
- See Certificates introduction for topics.

6.1.2. Managing certificates

Continue reading for information about how to refresh, replace, rotate, and list certificates.

- Refreshing a Red Hat Advanced Cluster Management webhook certificate
- Replacing certificates for alertmanager route
- Rotating the Gatekeeper webhook certificate

- Verifying certificate rotation
- Listing hub cluster managed certificates

6.1.2.1. Refreshing a Red Hat Advanced Cluster Management webhook certificate

You can refresh Red Hat Advanced Cluster Management managed certificates, which are certificates that are created and managed by Red Hat Advanced Cluster Management services.

Complete the following steps to refresh certificates managed by Red Hat Advanced Cluster Management:

1. Delete the secret that is associated with the Red Hat Advanced Cluster Management managed certificate by running the following command:

```
oc delete secret -n <namespace> <secret> 1
```

1 Replace **<namespace>** and **<secret>** with the values that you want to use.

2. Restart the services that are associated with the Red Hat Advanced Cluster Management managed certificate(s) by running the following command:

```
oc delete pod -n <namespace> -l <pod-label> 1
```

1 Replace **<namespace>** and **<pod-label>** with the values from the *Red Hat Advanced Cluster Management managed cluster certificates* table.

Note: If a **pod-label** is not specified, there is no service that must be restarted. The secret is recreated and used automatically.

6.1.2.2. Replacing certificates for alertmanager route

If you do not want to use the OpenShift default ingress certificate, replace observability alertmanager certificates by updating the alertmanager route. Complete the following steps:

1. Examine the observability certificate with the following command:

```
openssl x509  -noout -text -in ./observability.crt
```

2. Change the common name (**CN**) on the certificate to **alertmanager**.

3. Change the SAN in the **csr.cnf** configuration file with the hostname for your alertmanager route.

4. Create the two following secrets in the **open-cluster-management-observability** namespace. Run the following commands:

```
oc -n open-cluster-management-observability create secret tls alertmanager-byo-ca --cert
./ca.crt --key ./ca.key

oc -n open-cluster-management-observability create secret tls alertmanager-byo-cert --cert
./ingress.crt --key ./ingress.key
```

6.1.2.3. Replacing certificates for *rbac-query-proxy* route

You can replace certificates for the **rbac-query-proxy** route. See Bringing your own observability Certificate Authority (CA) certificates.

When you create a Certificate Signing Request (CSR) by using the **csr.cnf** file, update the **DNS.1** field in the **subjectAltName** section to match the host name of the **rbac-query-proxy** route.

Complete the following steps:

1. Run the following command to retrieve the host name:

```
oc get route rbac-query-proxy -n open-cluster-management-observability -o jsonpath="
{.spec.host}"
```

2. Run the following command to create a **proxy-byo-ca** secret by using the generated certificates:

```
oc -n open-cluster-management-observability create secret tls proxy-byo-ca --cert ./ca.crt --
key ./ca.key
```

3. Run the following command to create a **proxy-byo-cert** secret by using the generated certificates:

```
oc -n open-cluster-management-observability create secret tls proxy-byo-cert --cert
./ingress.crt --key ./ingress.key
```

6.1.2.4. Rotating the gatekeeper webhook certificate

Complete the following steps to rotate the gatekeeper webhook certificate:

1. Edit the secret that contains the certificate with the following command:

```
oc edit secret -n openshift-gatekeeper-system gatekeeper-webhook-server-cert
```

2. Delete the following content in the **data** section: **ca.crt**, **ca.key**, **tls.crt**, and **tls.key**.

3. Restart the gatekeeper webhook service by deleting the **gatekeeper-controller-manager** pods with the following command:

```
oc delete pod -n openshift-gatekeeper-system -l control-plane=controller-manager
```

The gatekeeper webhook certificate is rotated.

6.1.2.5. Verifying certificate rotation

Verify that your certificates are rotated using the following steps:

1. Identify the secret that you want to check.

2. Check the **tls.crt** key to verify that a certificate is available.

3. Display the certificate information by using the following command:

```
oc get secret <your-secret-name> -n open-cluster-management -o jsonpath='{.data.tls\.crt}' |
base64 -d | openssl x509 -text -noout
```

Replace **<your-secret-name>** with the name of secret that you are verifying. If it is necessary, also update the namespace and JSON path.

4. Check the **Validity** details in the output. View the following **Validity** example:

```
Validity
        Not Before: Jul 13 15:17:50 2023 GMT (1)
        Not After : Jul 12 15:17:50 2024 GMT (2)
```

(1) The **Not Before** value is the date and time that you rotated your certificate.

(2) The **Not After** value is the date and time for the certificate expiration.

6.1.2.6. Listing hub cluster managed certificates

You can view a list of hub cluster managed certificates that use OpenShift Service Serving Certificates service internally. Run the following command to list the certificates:

```
for ns in multicluster-engine open-cluster-management ; do echo "$ns:" ; oc get secret -n $ns -o
custom-
columns=Name:.metadata.name,Expiration:.metadata.annotations.service\\.beta\\.openshift\\.io/expiry
| grep -v '<none>' ; echo ""; done
```

For more information, see *OpenShift Service Serving Certificates* in the *Additional resources* section.

Note: If observability is enabled, there are additional namespaces where certificates are created.

6.1.2.7. Additional resources

- See Securing service traffic using service serving certificate secrets .
- See Certificates introduction.