Red Hat Advanced Cluster Management for Kubernetes 2.9

Governance

Governance

Last Updated: 2025-06-06

Red Hat Advanced Cluster Management for Kubernetes 2.9 Governance

Governance

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. 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

1.1. CERTIFICATES INTRODUCTION

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

- Certificates
- Bringing your own observability Certificate Authority (CA) certificates
- Managing certificates

1.2. 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.

1.2.1. 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 1.1. 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

Namespace	Secret name	Pod label	
app=multicluster-operators-application	open-cluster-management	multicluster-operators-application-svc-signed-ca	app=multicluster-operators-application
open-cluster-management-hub	registration-webhook-serving-cert signer-secret	Not required	open-cluster-management-hub

1.2.2. 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 1.2. 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

1.2.2.1. 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.

1.2.3. 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.

1.2.4. Bringing your own observability Certificate Authority (CA) certificates

When you install Red Hat Advanced Cluster Management for Kubernetes, only Certificate Authority (CA) certificates for observability are provided by default. If you do not want to use the default

observability CA certificates generated by Red Hat Advanced Cluster Management, you can choose to bring your own observability CA certificates before you enable observability.

1.2.4.1. Generating CA certificates by using OpenSSL commands

Observability requires two CA certificates, one for the server-side and the other is for the client-side.

- Generate your CA RSA private keys with the following commands:

```
openssl genrsa -out serverCAKey.pem 2048
openssl genrsa -out clientCAKey.pem 2048
```

- Generate the self-signed CA certificates using the private keys. Run the following commands:

```
openssl req -x509 -sha256 -new -nodes -key serverCAKey.pem -days 1825 -out
serverCACert.pem
openssl req -x509 -sha256 -new -nodes -key clientCAKey.pem -days 1825 -out
clientCACert.pem
```

1.2.4.2. Creating the secrets associated with the BYO observability CA certificates

Complete the following steps to create the secrets:

1. Create the **observability-server-ca-certs** secret by using your certificate and private key. Run the following command:

```
oc -n open-cluster-management-observability create secret tls observability-server-ca-certs -
-cert ./serverCACert.pem --key ./serverCAKey.pem
```

2. Create the **observability-client-ca-certs** secret by using your certificate and private key. Run the following command:

```
oc -n open-cluster-management-observability create secret tls observability-client-ca-certs --
cert ./clientCACert.pem --key ./clientCAKey.pem
```

1.2.4.3. Additional resources

- See Managing certificates.
- Return to the Certificates introduction.

1.2.5. 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

1.2.5.1. 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.

1.2.5.2. Replacing certificates for alertmanager route

You can replace alertmanager certificates by updating the alertmanager route, if you do not want to use the OpenShift default ingress certificate. 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
```

1.2.5.3. 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.

1.2.5.4. 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.

1.2.5.5. 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.

1.2.5.6. Additional resources

- OpenShift Service Serving Certificates
- Certificates introduction

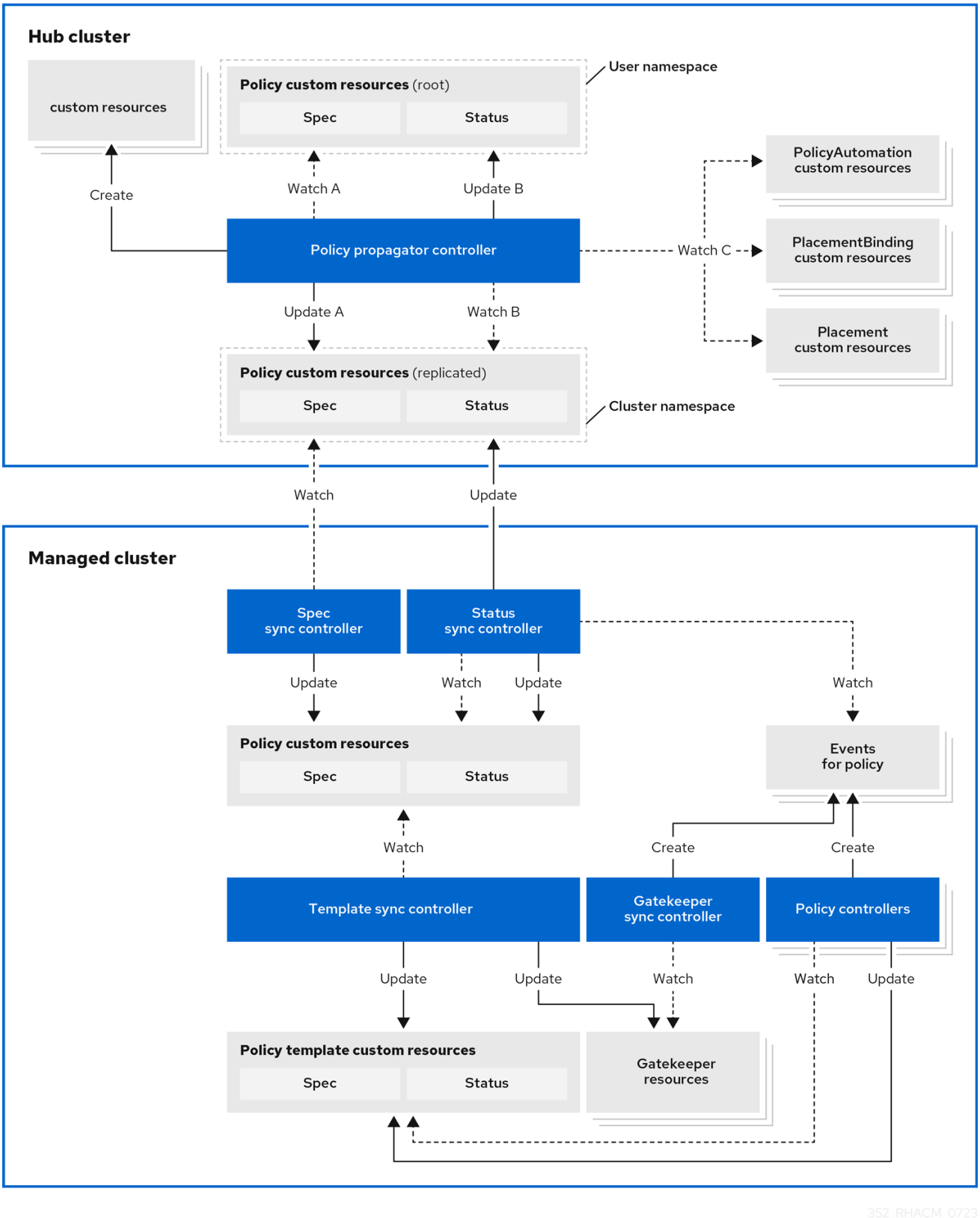
CHAPTER 2. GOVERNANCE

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:

- Governance architecture
- Policy overview
- Policy controllers introduction
- Policy controller advanced configuration
- Supported policies
- Policy dependencies
- Manage Governance dashboard
- Secure the hub cluster
- Integrate third-party policy controllers

2.1. GOVERNANCE 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:

- **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.

- **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.

2.1.1. 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.

2.1.2. Additional resources

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

2.2. POLICY OVERVIEW

Use the Red Hat Advanced Cluster Management for Kubernetes security policy framework to create and manage policies. Kubernetes custom resource definition instances are used to create policies.

Each Red Hat Advanced Cluster Management policy can have at least one or more templates. For more details about the policy elements, view the *Policy YAML table* section on this page.

The 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. For more on how to define a **Placement** resource see Placement overview in the Cluster lifecycle documentation.

Important:

- You must create the **PlacementBinding** to bind your policy with a **Placement** in order to propagate the policy to the managed clusters.

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

- You can create a policy in any namespace on the hub cluster except the cluster namespace. If you create a policy in the cluster namespace, it is deleted by Red Hat Advanced Cluster Management for Kubernetes.
- Each client and provider is responsible for ensuring that their managed cloud environment meets internal enterprise security standards for software engineering, secure engineering, resiliency, security, and regulatory compliance for workloads hosted on Kubernetes clusters.

Use the governance and security capability to gain visibility and remediate configurations to meet standards.

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

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

2.2.1. 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:
  dependencies:
  - apiVersion: policy.open-cluster-management.io/v1
    compliance:
    kind: Policy
    name:
    namespace:
  policy-templates:
    - objectDefinition:
        apiVersion:
        kind:
        metadata:
          name:
        spec:
  remediationAction:
  disabled:
---
apiVersion: apps.open-cluster-management.io/v1
kind: PlacementBinding
metadata:
bindingOverrides:
  remediationAction:
subFilter:
  name:
placementRef:
  name:
  kind:
  apiGroup:
subjects:
- name:
  kind:
  apiGroup:
---
apiVersion: apps.open-cluster-management.io/v1
kind: PlacementRule
metadata:
  name:
spec:
  clusterConditions:
  - type:
  clusterLabels:
    matchLabels:
      cloud:
```

2.2.2. 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.
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).

Field	Optional or required	Description
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.
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.
spec.policy-templates.ignorePending	Optional	Used to mark a policy template as compliant until the dependency criteria is verified.
spec.disabled	Required	Set the value to **true** or **false**. The **disabled** parameter provides the ability to enable and disable your policies.

Field	Optional or required	Description
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. **Important:** Some policy kinds might not support the enforce feature.

2.2.3. 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: PlacementRule
  apiGroup: apps.open-cluster-management.io
subjects:
- name: policy-role
  kind: Policy
  apiGroup: policy.open-cluster-management.io
---
apiVersion: apps.open-cluster-management.io/v1
kind: PlacementRule
metadata:
  name: placement-policy-role
spec:
  clusterConditions:
  - status: "True"
    type: ManagedClusterConditionAvailable
  clusterSelector:
    matchExpressions:
      - {key: environment, operator: In, values: ["dev"]}
```

2.2.4. Placement YAML sample file

The **PlacementBinding** and **Placement** resources can be combined with the previous policy example to deploy the policy using the cluster **Placement** API instead of the **PlacementRule** (Deprecated) API.

```
---
apiVersion: policy.open-cluster-management.io/v1
kind: PlacementBinding
metadata:
  name: binding-policy-role
bindingOverrides:
  remediationAction: null
subFilter: null
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
---
//Depends on if governance would like to use v1beta1
```

```
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"]}
```

- Refer to Policy controllers.
- See Managing security policies to create and update a policy. You can also enable and update Red Hat Advanced Cluster Management policy controllers to validate the compliance of your policies.
- Return to the Governance documentation.

2.3. POLICY CONTROLLERS INTRODUCTION

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
- IAM policy controller (Deprecated)
- Policy set 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.

2.3.1. Kubernetes configuration policy controller

The configuration policy controller can be used to configure any Kubernetes resource and apply security policies across your clusters. 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.

A Kubernetes object is defined (in whole or in part) in the **object-templates** array in the configuration policy, indicating to the configuration policy controller of the fields to compare with objects on the managed cluster. The configuration policy controller communicates with the local Kubernetes API server to get the list of your configurations that are in your cluster.

The configuration policy controller is created on the managed cluster during installation. The configuration policy controller supports the **enforce** and the **InformOnly** feature to remediate when the configuration policy is non-compliant.

When the **remediationAction** for the configuration policy is set to **enforce**, the controller applies the specified configuration to the target managed cluster.

When the **remediationAction** for the configuration policy 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**.

Note: Configuration policies that specify an object without a name can only be **inform**.

You can also use templated values within configuration policies. For more information, see *Template processing*.

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

2.3.1.1. Configuration policy sample

```
apiVersion: policy.open-cluster-management.io/v1
kind: ConfigurationPolicy
metadata:
  name: policy-config
spec:
  namespaceSelector:
    include: ["default"]
    exclude: []
    matchExpressions: []
    matchLabels: {}
  remediationAction: inform
  severity: low
  evaluationInterval:
    compliant:
    noncompliant:
  object-templates:
  - complianceType: musthave
    objectDefinition:
      apiVersion: v1
      kind: Pod
      metadata:
        name: pod
      spec:
        containers:
        - image: pod-image
          name: pod-name
          ports:
          - containerPort: 80
  - complianceType: musthave
    objectDefinition:
    apiVersion: v1
    kind: ConfigMap
    metadata:
      name: myconfig
      namespace: default
      data:
      testData: hello
    spec:
...
```

–

2.3.1.2. Configuration policy YAML table

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 **ConfigurationPolicy** to indicate the type of policy.
metadata.name	Required	The name of the policy.
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**.
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.compliant	Optional	Used to define how often the policy is evaluated when it is in the compliant state. The values must be in the format of a duration which is a sequence of numbers with time unit suffixes. For example, **12h30m5s** represents 12 hours, 30 minutes, and 5 seconds. It can also be set to **never** so that the policy is not reevaluated on the compliant cluster, unless the policy **spec** is updated. By default, the minimum time between evaluations for configuration policies is approximately 10 seconds when the **evaluationInterval.compliant** is not set or empty. This can be longer if the configuration policy controller is saturated on the managed cluster.
spec.evaluationInterval.noncompliant	Optional	Used to define how often the policy is evaluated when it is in the non-compliant state. Similar to the **evaluationInterval.compliant** parameter, the values must be in the format of a duration which is a sequence of numbers with time unit suffixes. It can also be set to **never** so that the policy is not reevaluated on the non-compliant cluster, unless the policy **spec** is updated.
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	Used to define the desired state of the Kubernetes object on the managed clusters. You must 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.
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.

2.3.1.3. Additional resources

See the following topics for more information:

- See the Policy overview 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.
- 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 Manage Governance dashboard.
- See Template processing.

2.3.2. 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 Policy overview 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**.

2.3.2.1. 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:
```

2.3.2.1.1. Certificate policy controller YAML table

Table 2.3. 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 **CertificatePolicy** to indicate the type of policy.
metadata.name	Required	The name to identify the policy.

Field	Optional or required	Description
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.
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**.

Field	Optional or required	Description
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.
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.

Field	Optional or required	Description
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.

2.3.2.2. 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.

2.3.2.3. 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.

2.3.3. IAM policy controller (Deprecated)

The Identity and Access Management (IAM) policy controller can be used to receive notifications about IAM policies that are non-compliant. The compliance check is based on the parameters that you configure in the IAM policy. The IAM 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. See the Policy YAML structure documentation for more details on the hub cluster policy.

The IAM policy controller monitors for the desired maximum number of users with a particular cluster role (i.e. **ClusterRole**) in your cluster. The default cluster role to monitor is **cluster-admin**. The IAM policy controller communicates with the local Kubernetes API server.

The IAM policy controller runs on your managed cluster. Continue reading to learn more.

2.3.3.1. IAM policy YAML structure

View the following example of an IAM policy and review the parameters in the YAML table:

```
apiVersion: policy.open-cluster-management.io/v1
kind: IamPolicy
metadata:
  name:
spec:
```

```
clusterRole:
severity:
remediationAction:
maxClusterRoleBindingUsers:
ignoreClusterRoleBindings:
```

2.3.3.2. IAM policy YAML table

View the following parameter table for descriptions:

Table 2.4. 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.
spec.clusterRole	Optional	The cluster role (i.e. **ClusterRole**) to monitor. This defaults to **cluster-admin** if not specified.
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.remediationAction	Optional	Specifies the remediation of your policy. Enter **inform**. The IAM policy controller only supports the **inform** feature.

Field	Optional or required	Description
spec.ignoreClusterRoleBindings	Optional	A list of regular expression (regex) values that indicate which cluster role binding names to ignore. These regular expression values must follow Go regexp syntax. By default, all cluster role bindings that have a name that starts with **system:** are ignored. It is recommended to set this to a stricter value. To not ignore any cluster role binding names, set the list to a single value of **.^** or some other regular expression that never matches.
spec.maxClusterRoleBindingUsers	Required	Maximum number of IAM role bindings that are available before a policy is considered non-compliant.

2.3.3.3. Additional resources

- See Managing security policies for more information.
- See Policy controllers for more topics.

2.3.4. 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.

Deprecated: PlacementRule

Note: The Red Hat Advanced Cluster Management hardening 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.

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

2.3.4.1. 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: apps.open-cluster-management.io
  kind: PlacementRule
  name: demo-policyset-pr
subjects:
- apiGroup: policy.open-cluster-management.io
  kind: PolicySet
  name: demo-policyset
---
apiVersion: apps.open-cluster-management.io
kind: PlacementRule
metadata:
  name: demo-policyset-pr
spec:
  clusterConditions:pagewidth:
  - status: "True"
    type: ManagedCLusterConditionAvailable
  clusterSelectors:
    matchExpressions:
      - key: name
        operator: In
        values:
          - local-cluster
```

2.3.4.2. Policy set table

View the following parameter table for descriptions:

Table 2.5. Parameter table

Field	Optional or required	Description
apiVersion	Required	Set the value to **policy.open-cluster-management.io/v1beta1**.

Field	Optional or required	Description
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.

2.3.4.3. 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
    placementRule: placement1
    policySet: policyset-ps
```

2.3.4.4. 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.
- Return to the beginning of this topic, Policy set controller.

2.4. 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, **governance-policy-framework**, **config-policy-controller**, **cert-policy-controller**, and **iam-policy-controller**.

Required access: Cluster administrator

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

2.4.1. Configure the concurrency

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 non-zero 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.

In the following YAML example, concurrency is set to **5** on the managed cluster that is 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
```

2.4.2. 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 non-zero 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.

In the following YAML example, 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
```

2.4.3. 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
```

2.4.4. Governance metric

The policy framework exposes metrics that show policy distribution and compliance. 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:

2.4.4.1. Metric: policy_governance_info

The **policy_governance_info** is collected by the OpenShift Container Platform monitoring feature, and some aggregate data is collected by Red Hat Advanced Cluster Management observability, if it is enabled.

Note: If observability is enabled, you can 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, as well as **PlacementRules** (deprecated) or **Placement**, and **PlacementBindings** to determine where to

create *propagated* policies in order to distribute the policy to managed clusters.

For both root and propagated policies, a metric of **0** is recorded if the policy is compliant, and **1** if it is non-compliant.

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 was 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 the metrics are not needed, and there are any concerns about performance or security, this feature can be disabled. 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.

2.4.4.2. 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.

2.4.5. Verify configuration changes

When the new configuration is applied by the controller, the **ManifestApplied** parameter is updated in the **ManagedClusterAddOn**. That condition timestamp can be used to 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
```

2.4.6. 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 .

2.5. SUPPORTED 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.

2.5.1. Table of sample configuration policies

View the following sample configuration policies:

Table 2.6. 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.

Policy sample	Description
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.
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.
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.

Policy sample	Description
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
- Red Hat OpenShift Platform Plus policy set

Refer to Governance for more topics.

2.5.2. 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

2.5.2.1. 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:
                ...
```

2.5.2.2. 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.

Field	Optional or required	Description
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.

2.5.2.3. Namespace policy sample

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

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

2.5.3. 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

2.5.3.1. 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:
                ...
```

2.5.3.2. Pod policy table

Table 2.7. 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.

Field	Optional or required	Description
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.

2.5.3.3. 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 Policy overview documentation to see a full description of the policy YAML structure and additional fields. Return to Managing configuration policies documentation to manage other policies.

2.5.4. 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

2.5.4.1. 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:
        ...
```

2.5.4.2. Memory usage policy table

Table 2.8. 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.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.

2.5.4.3. 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 Policy overview documentation, and to Kubernetes configuration policy controller to view other configuration policies that are monitored by the controller.

2.5.5. 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

2.5.5.1. 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:
                ...
```

2.5.5.2. Pod security policy table

Table 2.9. 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.

Field	Optional or required	Description
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.

2.5.5.3. Pod security policy sample

The support of pod security policies is removed from OpenShift Container Platform 4.12 and later, 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 Managing configuration policies for more information.
- Refer to the Policy overview 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.

2.5.6. 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. **Deprecated: PlacementRule**

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

- Role policy YAML structure

- Role policy table
- Role policy sample

2.5.6.1. 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:
                ...
---
apiVersion: policy.open-cluster-management.io/v1
kind: PlacementBinding
metadata:
  name: binding-policy-role
  namespace:
placementRef:
  name: placement-policy-role
  kind: PlacementRule
  apiGroup: apps.open-cluster-management.io
subjects:
```

```
- name: policy-role
  kind: Policy
  apiGroup: policy.open-cluster-management.io
---
apiVersion: apps.open-cluster-management.io/v1
kind: PlacementRule
metadata:
  name: placement-policy-role
  namespace:
spec:
  clusterConditions:
    - type: ManagedClusterConditionAvailable
      status: "True"
  clusterSelector:
    matchExpressions:
      []

          ...
```

2.5.6.2. Role policy table

Table 2.10. 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.

2.5.6.3. 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 Managing configuration policies for more information. See the Kubernetes configuration policy controller to view other configuration policies that are monitored the controller.

2.5.7. 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

2.5.7.1. 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:
    ...
```

2.5.7.2. Role binding 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.

2.5.7.3. 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 Policy overview documentation . Refer to Kubernetes configuration policy controller documentation to learn about other configuration policies.

2.5.8. 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

2.5.8.1. 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:
        ...
```

2.5.8.2. 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.

2.5.8.3. 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 (SCC) .

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

2.5.9. 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

2.5.9.1. 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:
                ...
```

2.5.9.2. ETCD encryption policy table

Table 2.11. 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 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.

2.5.9.3. ETCD encryption policy sample

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

2.5.10. 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.

2.5.10.1. 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.

2.5.10.2. 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**.

2.5.10.3. 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.

2.5.11. 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.

2.5.11.1. 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.

2.5.12. 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.

2.5.12.1. 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.

2.5.13. 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

2.5.13.1. 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
```

2.5.13.2. 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.

2.5.14. 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.

2.5.14.1. Prerequisites

- Install Red Hat OpenShift Container Platform 4.12 or later, on Amazon Web Services (AWS) environment.
- Install Red Hat Advanced Cluster Management for Kubernetes 2.7 or later.
- Install the Policy Generator Kustomize plugin. See the Policy Generator documentation for more information.

2.5.14.2. 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 2.12. 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.

2.5.14.3. 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.

2.6. MANAGE 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.

2.6.1. 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.

2.6.2. 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.

2.6.3. Additional resources

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

- Managing security policies
- Managing configuration policies
- Managing gatekeeper policies
- Configuring Ansible Automation Platform for governance
- Governance

2.6.4. 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

2.6.4.1. Prerequisites

- Red Hat OpenShift Container Platform 4.12 or later
- 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.

2.6.4.2. 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.

2.6.4.3. 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.

2.7. TEMPLATE PROCESSING INTRODUCTION

Configuration policies support the inclusion of Golang text templates in the object definitions. 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 configuration policies with dynamic content, and inform or enforce Kubernetes resources that are customized to the target cluster.

A configuration 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. On the managed cluster, the **ConfigurationPolicyController** processes any managed cluster templates in the policy definition and then enforces or verifies the fully resolved object definition.

The template syntax must be conformed 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.

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

2.7.1. Comparison of hub cluster and managed cluster templates

Table 2.13. 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
Access control	You can only reference namespaced Kubernetes objects that are in the same namespace as the **Policy** resource.	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 read access with **ConfigurationPolicy** resource objects on the managed clusters. Additionally, if etcd encryption is enabled, the **Policy** and **ConfigurationPolicy** 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.
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 in the **ConfigurationPolicyController** on the managed cluster. Policies are processed periodically, which automatically updates the resolved object definition with data in the referenced resources.
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

2.7.2. Template functions

Template functions, such as resource-specific and generic **lookup** template functions, are available for referencing Kubernetes resources on the hub cluster (using the **{{hub ... hub}}** delimiters), or on the managed cluster (using the **{{ ... }}** delimiters). See *Template processing* for more details. The resource-specific functions are used for convenience and makes content of the resources more

accessible. 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 also available.

To conform templates with YAML syntax, templates must be set 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 descriptions and examples for some of the custom template functions that are supported:

- *fromSecret* function
- *fromConfigMap* function
- *fromClusterClaim* function
- *lookup* function
- *base64enc* function
- *base64dec* function
- *indent* function
- *autoindent* function
- *toInt* function
- *toBool* function
- *protect* function
- *toLiteral* function
- *copySecretData* function
- *copyConfigMapData* function
- Supported Sprig open source functions

2.7.2.1. *fromSecret* function

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.

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

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. View the following configuration policy that enforces a **Secret** resource on the target cluster. The value for the

PASSWORD data key is a template that references the secret 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:
        USER_NAME: YWRtaW4=
        PASSWORD: '{{ fromSecret "default" "localsecret" "PASSWORD" }}'
      kind: Secret
      metadata:
        name: demosecret
        namespace: test
      type: Opaque
  remediationAction: enforce
  severity: low
```

2.7.2.2. *fromConfigmap* function

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

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

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. You receive a policy violation if the Kubernetes **ConfigMap** 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. View the following configuration policy that enforces a Kubernetes resource on the target managed cluster. The value for the **log-file** data key is a template that retrieves the value of the **log-file** from the ConfigMap, **logs-config** from the **default** namespace, and the **log-level** is set to the data key **log-level**.

```
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
      apiVersion: v1
      metadata:
        name: demo-app-config
        namespace: test
      data:
        app-name: sampleApp
        app-description: "this is a sample app"
        log-file: '{{ fromConfigMap "default" "logs-config" "log-file" }}'
        log-level: '{{ fromConfigMap "default" "logs-config" "log-level" }}'
  remediationAction: enforce
  severity: low
```

2.7.2.3. *fromClusterClaim* function

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)
```

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

```
apiVersion: policy.open-cluster-management.io/v1
kind: ConfigurationPolicy
metadata:
  name: demo-clusterclaims
  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:
        # Configuration values can be set as key-value properties
        platform: '{{ fromClusterClaim "platform.open-cluster-management.io" }}'
        product: '{{ fromClusterClaim "product.open-cluster-management.io" }}'
```

```
      version: '{{ fromClusterClaim "version.openshift.io" }}'
  remediationAction: enforce
  severity: low
```

2.7.2.4. *lookup* function

The **lookup** function returns the Kubernetes resource as a JSON compatible map. 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 *Supported Sprig open source functions* section.

View the following syntax for the function:

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

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. 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. 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:

```
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:
        # Configuration values can be set as key-value properties
        app-name: sampleApp
        app-description: "this is a sample app"
        metrics-url: |
          http://{{ (lookup "v1" "Service" "default" "metrics").spec.clusterIP }}:8080
  remediationAction: enforce
  severity: low
```

2.7.2.5. *base64enc* function

The **base64enc** function returns a **base64** encoded value of the input **data string**. View the following syntax for the function:

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

When you use this function, enter a string value. 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 }}'
```

2.7.2.6. *base64dec* function

The **base64dec** function returns a **base64** decoded value of the input **enc-data string**. View the following syntax for the function:

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

When you use this function, enter a string value. 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 }}"
```

2.7.2.7. *indent* function

The **indent** function returns the padded **data string**. View the following syntax for the function:

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

When you use this function, enter a data string with the specific number of spaces. 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
}}
```

2.7.2.8. *autoindent* function

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 }}
```

2.7.2.9. *toInt* function

The **toInt** function casts and returns the integer value of the input value. Also, 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. View the following syntax for the function:

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

When you use this function, enter the data that needs to be casted as an integer. 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 }}
```

2.7.2.10. *toBool* function

The **toBool** function converts the input string into a boolean, and returns the boolean. Also, 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. View the following syntax for the function:

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

When you use this function, enter the string data that needs to be converted to a boolean. 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 }}
```

2.7.2.11. *protect* function

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.

2.7.2.12. *toLiteral* function

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 ConfigMap 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"]
```

2.7.2.13. *copySecretData* function

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" }}'
```

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

2.7.2.14. *copyConfigMapData* function

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

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

2.7.2.15. Supported Sprig open source functions

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

Table 2.14. 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**

Sprig library	Functions
Dictionaries and dict	**dig**
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**, **quote**, **replace**, **split**, **splitn**, **substr**, **trim**, **trimAll**, **trunc**, **upper**
Version comparison	**semver**, **semverCompare**

2.7.2.16. Additional resources

- Return to Template processing
- 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.

2.7.3. 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

2.7.3.1. 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.

2.7.3.2. 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 }}
```

2.7.3.3. 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**.

2.7.3.4. Additional resources

- See Template functions for more details.
- Return to Template processing.
- See Kubernetes configuration policy controller for more details.
- Also refer to the Red Hat OpenShift Container Platform etcd encryption documentation .

2.8. 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

2.8.1. 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. Enter a valid value for the *Cluster selector* field to define a **Placement** and **PlacementBinding**.

See Resources that support support set-based requirements in the Kubernetes documentation for a valid expression. View the definitions of the objects that are required for your Red Hat Advanced Cluster Management policy:

- *PlacementBinding*: Binds the placement.

View more descriptions of the policy YAML files in the Policy overview.

2.8.1.1. 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** to bind your policy to your **PlacementRule**(Deprecated). Your **PlacementBinding** might resemble the following YAML sample:

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

2.8.1.1.1. 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>
```

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

See the following **PlacementRule** example:

```
apiVersion: apps.open-cluster-management.io/v1
 kind: PlacementRule
 metadata:
   name: placement-pod
spec:
  clusterConditions: []
  clusterSelector:
     matchLabels:
       cloud: "IBM"
```

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

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

2.8.1.2.1. 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.

2.8.1.3. 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>
```

2.8.1.4. 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.
5. View the stable **Policysets**, which require the Policy Generator for deployment, PolicySets--Stable.

2.8.2. Updating security policies

Learn to update security policies.

2.8.2.1. 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 **PlacementRule** are both created.

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

2.8.2.2. 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.

2.8.2.3. 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.

2.8.3. Deleting a security policy

Delete a security policy from the CLI or the console.

- Delete a security policy from the CLI:
  a. Delete a security policy by running the following command:

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

  After your policy is deleted, it is removed from your target cluster or clusters. Verify that your policy is removed by running the following command: **oc get policies.policy.open-cluster-management.io <policy-name> -n <policy-namespace>**

- Delete a security policy from the console:
  From the navigation menu, click **Governance** to view a table list of your policies. Click the **Actions** icon for the policy you want to delete in the policy violation table.

  Click **Remove**. From the *Remove policy* dialog box, click **Remove policy**.

2.8.3.1. 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.

To manage other policies, see Managing security policies for more information. Refer to Governance for more topics about policies.

2.8.4. 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.

2.8.5. Managing configuration policies

Learn to create, apply, view, and update your configuration policies.

Required access: Administrator or cluster administrator

- Creating a configuration policy
- Updating configuration policies
- Deleting a configuration policy

2.8.5.1. Creating a configuration policy

You can create a YAML file for your configuration policy from the command line interface (CLI) or from the console.

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. View the following sections to create a configuration policy:

2.8.5.1.1. 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.

2.8.5.1.2. 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>
```

2.8.5.1.3. Creating a configuration policy from the console

As you create a configuration policy from the console, a YAML file is also created in the YAML editor.

1. Log in to your cluster from the console, and select **Governance** from the navigation menu.

2. Click **Create policy**. Specify the policy you want to create by selecting one of the configuration policies for the specification parameter.

3. Continue with configuration policy creation by completing the policy form. Enter or select the appropriate values for the following fields:

- Name
- Specifications
- Cluster selector
- Remediation action
- Standards
- Categories
- Controls

4. Click **Create**. Your configuration policy is created.

2.8.5.1.4. 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.

2.8.5.2. Updating configuration policies

Learn to update configuration policies by viewing the following section.

2.8.5.2.1. Disabling configuration policies

Disable your configuration policy. Similar to the instructions mentioned earlier, log in and navigate to the *Governance* page to complete the tasks.

1. Select the **Actions** icon for a configuration policy from the table list, then click **Disable**. The *Disable Policy* dialog box appears.
2. Click **Disable policy**.

The policy is disabled, but not deleted.

2.8.5.3. Deleting a configuration policy

Delete a configuration policy from the CLI or the console.

- Delete a configuration policy from the CLI with the following procedure:
  1. Run the following command to delete the policy from your target cluster or clusters:

     ```
     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>
```

- Delete a configuration policy from the console with the following procedure:

  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, then click **Remove**.
  3. From the *Remove policy* dialog box, click **Remove policy**.

Your policy is deleted.

2.8.5.4. 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. For details to manage other policies, refer to Managing security policies.

2.8.6. Managing Gatekeeper operator policies

Use the Gatekeeper operator policy to install the Gatekeeper operator and Gatekeeper on a managed cluster. Learn to create, view, and update your Gatekeeper operator policies in the following sections.

Required access: Cluster administrator

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

2.8.6.1. Installing Gatekeeper using a Gatekeeper operator policy

Use the governance framework to install the Gatekeeper operator. Gatekeeper operator is available in the OpenShift Container Platform catalog. See *Adding Operators to a cluster* in the OpenShift Container Platform documentation for more information.

Use the configuration policy controller to install the Gatekeeper operator policy. During the install, the operator group and subscription pull the Gatekeeper operator to install it in your managed cluster. Then, the Gatekeeper operator creates a Gatekeeper custom resource to configure Gatekeeper. View the Gatekeeper operator custom resource sample.

Gatekeeper operator policy is monitored by the Red Hat Advanced Cluster Management configuration policy controller, where **enforce** remediation action is supported. Gatekeeper operator policies are created automatically by the controller when set to **enforce**.

2.8.6.2. Creating a Gatekeeper policy from the console

Install the Gatekeeper policy by creating a Gatekeeper policy from the console. Alternatively, navigate to the *Additional resources* section for a reference to the sample YAML to deploy **policy-gatekeeper-operator.yaml**.

After you log in to your cluster, navigate to the *Governance* page.

Select **Create policy**. As you complete the form, select **Gatekeeper Operator** from the *Specifications* field. The parameter values for your policy are automatically populated and the policy is set to **inform** by default. Set your remediation action to **enforce** to install Gatekeeper.

Note: Default values are generated by the operator.

2.8.6.2.1. Gatekeeper operator custom resource

```
apiVersion: operator.gatekeeper.sh/v1alpha1
kind: Gatekeeper
metadata:
  name: gatekeeper
spec:
  audit:
    replicas: 1
    logLevel: DEBUG
    auditInterval: 10s
    constraintViolationLimit: 55
    auditFromCache: Enabled
    auditChunkSize: 66
    emitAuditEvents: Enabled
    resources:
      limits:
        cpu: 500m
        memory: 150Mi
      requests:
        cpu: 500m
        memory: 130Mi
  validatingWebhook: Enabled
  webhook:
    replicas: 2
    logLevel: ERROR
    emitAdmissionEvents: Enabled
    failurePolicy: Fail
    resources:
      limits:
        cpu: 480m
        memory: 140Mi
      requests:
        cpu: 400m
        memory: 120Mi
  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"
```

2.8.6.3. 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 **installPlanApproval**. The operator upgrades automatically when **installPlanApproval** is set to **Automatic**.

You must approve the upgrade of the Gatekeeper operator manually, for each cluster, when **installPlanApproval** is set to **Manual**.

2.8.6.4. Updating Gatekeeper operator policy

Learn to update the Gatekeeper operator policy by viewing the following section.

2.8.6.4.1. Viewing Gatekeeper operator policy from the console

View your Gatekeeper operator policy and its status from the console.

After you log in to your cluster from the console, click **Governance** 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.

Select the **policy-gatekeeper-operator** policy to view more details. View the policy violations by selecting the *Clusters* tab.

2.8.6.4.2. Disabling Gatekeeper operator policy

Disable your gatekeeper operator policy.

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 for the **policy-gatekeeper-operator** policy, then click **Disable**. The *Disable Policy* dialog box appears.

Click **Disable policy**. Your **policy-gatekeeper-operator** policy is disabled.

2.8.6.5. Deleting Gatekeeper operator policy

Delete the Gatekeeper operator policy from the CLI or the console.

- Delete Gatekeeper operator policy from the CLI:

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

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

  After your policy is deleted, it is removed from your target cluster or clusters.

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

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

- Delete Gatekeeper operator policy from the console:
  Navigate to the *Governance* page to view a table list of your policies.

  Similar to the previous console instructions, click the **Actions** icon for the **policy-gatekeeper-operator** policy. Click **Remove** to delete the policy. From the *Remove policy* dialog box, click **Remove policy**.

Your Gatekeeper operator policy is deleted.

2.8.6.6. Uninstalling Gatekeeper policy, Gatekeeper, and Gatekeeper operator policy

Complete the following steps to uninstall Gatekeeper policy, Gatekeeper, and Gatekeeper operator policy:

1. Remove the Gatekeeper **Constraint** and **ConstraintTemplate** that is applied on your managed cluster:

   a. Edit your Gatekeeper operator policy. Locate the **ConfigurationPolicy** template that you used to create the Gatekeeper **Constraint** and **ConstraintTemplate**.

   b. Change the value for **complianceType** of the **ConfigurationPolicy** template to **mustnothave**.

   c. Save and apply the policy.

2. Remove Gatekeeper instance from your managed cluster:

   a. Edit your Gatekeeper operator policy. Locate the **ConfigurationPolicy** template that you used to create the Gatekeeper custom resource.

   b. Change the value for **complianceType** of the **ConfigurationPolicy** template to **mustnothave**.

3. Remove the Gatekeeper operator that is on your managed cluster:

   a. Edit your Gatekeeper operator policy. Locate the **ConfigurationPolicy** template that you used to create the Subscription CR.

   b. Change the value for **complianceType** of the **ConfigurationPolicy** template to **mustnothave**.

Gatekeeper policy, Gatekeeper, and Gatekeeper operator policy are uninstalled.

2.8.6.7. Additional resources

- See Integrating Gatekeeper constraints and constraint templates for details about Gatekeeper.
- See the Policy Gatekeeper sample.
- See Gatekeeper Helm Chart for an explanation of the optional parameters that can be used for the Gatekeeper operator policy.

- For a list of topics to integrate third-party policies with the product, see Integrate third-party policy controllers.

2.8.7. 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**

2.8.8. 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.

2.8.8.1. 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: PlacementRule
  apiGroup: apps.open-cluster-management.io
subjects:
- name: policy-configure-subscription-admin-hub
  kind: Policy
  apiGroup: policy.open-cluster-management.io
---
apiVersion: apps.open-cluster-management.io/v1
kind: PlacementRule
metadata:
  name: placement-policy-configure-subscription-admin-hub
spec:
  clusterConditions:
  - status: "True"
    type: ManagedClusterConditionAvailable
  clusterSelector:
    matchExpressions:
      - {key: name, operator: In, values: ["local-cluster"]}
```

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.

2.8.8.2. 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.

2.8.8.3. 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

2.9. POLICY DEPENDENCIES

Dependencies can be used to activate a policy or policy template when the dependency criteria are satisfied. The following fields are checked on the managed cluster, **dependencies** and **extraDependencies**. When a dependency is not met, the template status of the replicated policy template displays more details.

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:
  - apiVersion: policy.open-cluster-management.io/v1
    compliance: Compliant
    kind: Policy
    name: upstream-compliance-operator
    namespace: default
  disabled: false
  policy-templates:
  - 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
```

Note: A dependency cannot be used to apply a policy on one cluster based on the status of a policy in another cluster.

2.10. SECURE 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 OpenShift Container Platform security and compliance.
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

2.11. INTEGRATE THIRD-PARTY POLICY CONTROLLERS

Integrate third-party policies to create custom annotations within the policy templates to specify one or more compliance standards, control categories, and controls.

You can also use the third-party party policies from the policy-collection/community.

Learn to integrate the following third-party policies:

- Integrating Gatekeeper constraints and constraint templates
- Policy Generator

- Generating a policy to install an Operator

2.11.1. Integrating Gatekeeper constraints and constraint templates

Gatekeeper is a validating webhook with auditing capabilities that can enforce custom resource definition-based policies that are run with the Open Policy Agent (OPA). You can install Gatekeeper on your cluster by using the Gatekeeper operator policy. Gatekeeper constraints can be used to evaluate Kubernetes resource compliance. You can leverage OPA as the policy engine, and use Rego as the policy language.

Prerequisite: A Red Hat Advanced Cluster Management for Kubernetes or Red Hat OpenShift Container Platform Plus subscription is required to install Gatekeeper and apply Gatekeeper policies to your cluster. Gatekeeper is supported only on OpenShift Container Platform versions, except version 3.11, supported by the latest version of Red Hat Advanced Cluster Management.

Gatekeeper policies are written using constraint templates (**ConstraintTemplates**) and constraints. 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
    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 same as for other Red Hat Advanced Cluster Management policy types: **low**, **medium**, **high**, or **critical**.

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)**. Once a policy containing Gatekeeper constraints or **ConstraintTemplates** is deleted, the constraints and **ConstraintTemplates** are also deleted from the managed cluster.

To view the Gatekeeper audit results for a specific managed cluster from the console, navigate 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 the audit results are generated by Gatekeeper version 3.9 or newer.
- The Gatekeeper audit functionality 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:

```
apiVersion: policy.open-cluster-management.io/v1
kind: ConfigurationPolicy
metadata:
  name: policy-gatekeeper-admission
spec:
  remediationAction: inform # will be overridden by remediationAction in parent policy
  severity: low
  object-templates:
    - complianceType: mustnothave
      objectDefinition:
        apiVersion: v1
        kind: Event
        metadata:
          namespace: openshift-gatekeeper-system # set it to the actual namespace where
gatekeeper is running if different
          annotations:
            constraint_action: deny
            constraint_kind: K8sRequiredLabels
            constraint_name: ns-must-have-gk
            event_type: violation
```

2.11.1.1. Additional resources

- See **policy-gatekeeper-operator.yaml** for more details.
- For more details, see What is OPA Gatekeeper.
- See Managing configuration policies for more information about managing other policies.
- Refer to Governance for more topics on the security framework.

2.11.2. Policy Generator

The Policy Generator is a part of the Red Hat Advanced Cluster Management for Kubernetes application lifecycle subscription 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:

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

2.11.2.1. Policy Generator capabilities

The Policy Generator and its integration with the Red Hat Advanced Cluster Management application lifecycle subscription 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.

2.11.2.2. 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.

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

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

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:

```
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
```

The **configmap.yaml** 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 example:

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

The generated **Policy**, along with the generated **PlacementRule** and **PlacementBinding** might resemble the following example:

```
apiVersion: apps.open-cluster-management.io/v1
kind: PlacementRule
metadata:
  name: placement-config-data
  namespace: policies
spec:
  clusterConditions:
  - status: "True"
    type: ManagedClusterConditionAvailable
  clusterSelector:
    matchExpressions: []
---
apiVersion: policy.open-cluster-management.io/v1
kind: PlacementBinding
metadata:
  name: binding-config-data
  namespace: policies
placementRef:
  apiGroup: apps.open-cluster-management.io
  kind: PlacementRule
  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
```

2.11.2.3. Policy Generator configuration reference table

Note that 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 2.15. 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.
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**.

Field	Optional or required	Description
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**.
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 (**{}**).

Field	Optional or required	Description
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.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**.
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.

Field	Optional or required	Description
policyDefaults.evaluationInterval	Optional	Use the parameters **compliant** and **noncompliant** to specify the frequency for a policy to be evaluated when in a particular compliance state. When managed clusters have low CPU resources, the evaluation interval can be increased to reduce CPU usage on the Kubernetes API. 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 has become a particular compliance state.
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.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**.
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**.

Field	Optional or required	Description
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.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.
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**.

Field	Optional or required	Description
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**.
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**.

Field	Optional or required	Description
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.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 placement rules that contain the same cluster selectors.

Field	Optional or required	Description
policyDefaults.placement.labelSelector	Optional	Specify a placement rule by defining a cluster 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 rule 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.
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**.

Field	Optional or required	Description
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.
policies[].name	Required	The name of the policy to create.

Field	Optional or required	Description
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** and **metadataComplianceType** 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.
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.
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.

Field	Optional or required	Description
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.

2.11.2.4. 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 Gatekeeper for more details.
- Refer to the Kustomize documentation.
- Return to the Integrate third-party policy controllers documentation.