Red Hat Advanced Cluster Management for Kubernetes 2.2

Security

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

Last Updated: 2022-05-18

Red Hat Advanced Cluster Management for Kubernetes 2.2 Security

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

Abstract

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

Table of Contents

CHAPTER 1. SECURITY

Manage your security and role-based access control (RBAC) of 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 to onboard workloads to Identity and Access Management (IAM). For more information see, *Understanding authentication* in Understanding authentication in the OpenShift Container Platform documentation.

Review the following topics to learn more about securing your cluster:

- Role-based access control
- Credentials
- Certificates
- Governance and risk

1.1. ROLE-BASED ACCESS CONTROL

Red Hat Advanced Cluster Management for Kubernetes supports role-based access control (RBAC). Your role determines the actions that you can perform. RBAC is based on the authorization mechanisms in Kubernetes, similar to Red Hat OpenShift Container Platform. For more information about RBAC, see the OpenShift *RBAC* overview in the OpenShift Container Platform documentation.

Note: Action buttons are disabled from the console if the user-role access is impermissible.

View the following sections for details of supported RBAC by component:

- Overview of roles
- RBAC implementation
- Cluster lifecycle RBAC
- Application lifecycle RBAC
- Governance lifecycle RBAC
- Observability RBAC

1.1.1. Overview of roles

Some product resources are cluster-wide and some are namespace-scoped. You must apply cluster role bindings and namespace role bindings to your users for consistent access controls. View the table list of the following role definitions that are supported in Red Hat Advanced Cluster Management for Kubernetes:

Table 1.1. Role definition table

Role	Definition
cluster-admin	A user with cluster-wide binding to the **cluster-admin** role is an OpenShift Container Platform super user, who has all access.
open-cluster-management:cluster-manager-admin	A user with cluster-wide binding to the **cluster-manager-admin** role is a Red Hat Advanced Cluster Management for Kubernetes super user, who has all access. This role allows the user to create a ManagedCluster resource.
open-cluster-management:managed-cluster-x (admin)	A user with cluster binding to the **managed-cluster-x** role has administrator access to **managedcluster “X”** resource.
open-cluster-management:managed-cluster-x (viewer)	A user with cluster-wide binding to the **managed-cluster-x** role has view access to**managedcluster “X”** resource.
open-cluster-management:subscription-admin	A user with the **subscription-admin** role can create Git subscriptions that deploy resources to multiple namespaces. The resources are specified in Kubernetes resource YAML files in the subscribed Git repository. **Note**: When a non-subscription-admin user creates a subscription, all resources are deployed into the subscription namespace regardless of specified namespaces in the resources. For more information, see the Application lifecycle RBAC section.
admin, edit, view	Admin, edit, and view are OpenShift Container Platform default roles. A user with a namespace-scoped binding to these roles has access to **open-cluster-management** resources in a specific namespace, while cluster-wide binding to the same roles gives access to all of the **open-cluster-management** resources cluster-wide.

Important:

- Any user can create projects from OpenShift Container Platform, which gives administrator role permissions for the namespace.
- If a user does not have role access to a cluster, the cluster name is not visible. The cluster name is displayed with the following symbol: **-**.

1.1.2. RBAC implementation

RBAC is validated at the console level and at the API level. Actions in the console can be enabled or disabled based on user access role permissions. View the following sections for more information on RBAC for specific lifecycles in the product.

1.1.2.1. Cluster lifecycle RBAC

View the following cluster lifecycle RBAC operations.

To create and administer all managed clusters:

- Create a cluster role binding to the cluster role **open-cluster-management:cluster-manager-admin**. This role is a super user, which has access to all resources and actions. This role allows you to create cluster-scoped **managedcluster** resources, the namespace for the resources that manage the managed cluster, and the resources in the namespace. This role also allows access to provider connections and to bare metal assets that are used to create managed clusters.

  ```
  oc create clusterrolebinding <role-binding-name> --clusterrole=open-cluster-
  management:cluster-manager-admin
  ```

To administer a managed cluster named **cluster-name**:

- Create a cluster role binding to the cluster role **open-cluster-management:admin:<cluster-name>**. This role allows read/write access to the cluster-scoped **managedcluster** resource. This is needed because the **managedcluster** is a cluster-scoped resource and not a namespace-scoped resource.

  ```
  oc create clusterrolebinding (role-binding-name) --clusterrole=open-cluster-
  management:admin:<cluster-name>
  ```

- Create a namespace role binding to the cluster role **admin**. This role allows read/write access to the resources in the namespace of the managed cluster.

  ```
  oc create rolebinding <role-binding-name> -n <cluster-name> --clusterrole=admin
  ```

To view a managed cluster named **cluster-name**:

- Create a cluster role binding to the cluster role **open-cluster-management:view:<cluster-name>**. This role allows read access to the cluster-scoped **managedcluster** resource. This is needed because the **managedcluster** is a cluster-scoped resource and not a namespace-scoped resource.

  ```
  oc create clusterrolebinding <role-binding-name> --clusterrole=open-cluster-
  management:view:<cluster-name>
  ```

- Create a namespace role binding to the cluster role **view**. This role allows read-only access to the resources in the namespace of the managed cluster.

  ```
  oc create rolebinding <role-binding-name> -n <cluster-name> --clusterrole=view
  ```

View the following console and API RBAC tables for cluster lifecycle:

Table 1.2. Console RBAC table for cluster lifecycle

Action	Admin	Edit	View
Clusters	read, update, delete	read, update	read
Provider connections	create, read, update, and delete	create, read, update, and delete	read
Bare metal asset	create, read, update, delete	read, update	read

Table 1.3. API RBAC table for cluster lifecycle

API	Admin	Edit	View
managedclusters.cluster.open-cluster-management.io	create, read, update, delete	read, update	read
baremetalassets.inventory.open-cluster-management.io	create, read, update, delete	read, update	read
klusterletaddonconfigs.agent.open-cluster-management.io	create, read, update, delete	read, update	read
managedclusteractions.action.open-cluster-management.io	create, read, update, delete	read, update	read
managedclusterviews.view.open-cluster-management.io	create, read, update, delete	read, update	read
managedclusterinfos.internal.open-cluster-management.io	create, read, update, delete	read, update	read
manifestworks.work.open-cluster-management.io	create, read, update, delete	read, update	read

1.1.2.2. Application lifecycle RBAC

When you create an application, the ***subscription*** namespace is created and the configuration map is created in the ***subscription*** namespace. You must also have access to the ***channel*** namespace. When you want to apply a subscription, you must be a subscription administrator. For more information on managing applications, see Creating and managing subscriptions.

To perform application lifecycle tasks, users with the **admin** role must have access to the ***application*** namespace where the application is created, and to the ***managed cluster*** namespace. For example, the required access to create applications in namespace "N" is a namespace-scoped binding to the **admin** role for namespace "N".

View the following console and API RBAC tables for Application lifecycle:

Table 1.4. Console RBAC table for Application lifecycle

Action	Admin	Edit	View
Application	create, read, update, delete	create, read, update, delete	read
Channel	create, read, update, delete	create, read, update, delete	read
Subscription	create, read, update, delete	create, read, update, delete	read
Placement rule	create, read, update, delete	create, read, update, delete	read

Table 1.5. API RBAC table for application lifecycle

API	Admin	Edit	View
applications.app.k8s.io	create, read, update, delete	create, read, update, delete	read
channels.apps.open-cluster-management.io	create, read, update, delete	create, read, update, delete	read
deployables.apps.open-cluster-management.io	create, read, update, delete	create, read, update, delete	read
helmreleases.apps.open-cluster-management.io	create, read, update, delete	create, read, update, delete	read
placementrules.apps.open-cluster-management.io	create, read, update, delete	create, read, update, delete	read
subscriptions.apps.open-cluster-management.io	create, read, update, delete	create, read, update, delete	read
configmaps	create, read, update, delete	create, read, update, delete	read
secrets	create, read, update, delete	create, read, update, delete	read

API	Admin	Edit	View
namespaces	create, read, update, delete	create, read, update, delete	read

1.1.2.3. Governance lifecycle RBAC

To perform governance lifecycle operations, users must have access to the namespace where the policy is created, along with access to the **managedcluster** namespace where the policy is applied.

View the following examples:

- To view policies in namespace "N" the following role is required:
  - A namespace-scoped binding to the **view** role for namespace "N".
- To create a policy in namespace "N" and apply it on **managedcluster** "X", the following roles are required:
  - A namespace-scoped binding to the **admin** role for namespace "N".
  - A namespace-scoped binding to the **admin** role for namespace "X".

View the following console and API RBAC tables for Governance lifecycle:

Table 1.6. Console RBAC table for governance lifecycle

Action	Admin	Edit	View
Policies	create, read, update, delete	read, update	read
PlacementBindings	create, read, update, delete	read, update	read
PlacementRules	create, read, update, delete	read, update	read

Table 1.7. API RBAC table for Governance lifecycle

API	Admin	Edit	View
policies.policy.open-cluster-management.io	create, read, update, delete	read, update	read
placementbindings.policy.open-cluster-management.io	create, read, update, delete	read, update	read

1.1.2.4. Observability RBAC

To view the observability metrics for a managed cluster, you must have **view** access to that managed cluster on the hub cluster. View the following list of observability features:

- Access managed cluster metrics.
  Users are denied access to managed cluster metrics, if they are not assigned to the **view** role for the managed cluster on the hub cluster.
- Search for resources.

To view observability data in Grafana, you must have a **RoleBinding** resource in the same namespace of the managed cluster. View the following **RoleBinding** example:

```
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
 name: <replace-with-name-of-rolebinding>
 namespace: <replace-with-name-of-managedcluster-namespace>
subjects:
 - kind: <replace with User|Group|ServiceAccount>
   apiGroup: rbac.authorization.k8s.io
   name: <replace with name of User|Group|ServiceAccount>
roleRef:
 apiGroup: rbac.authorization.k8s.io
 kind: ClusterRole
 name: view
```

See Role binding policy for more information. See Customizing observability to configure observability.

- Use the Visual Web Terminal if you have access to the managed cluster.

To create, update, and delete the **MultiClusterObservability** custom resource. View the following RBAC table:

Table 1.8. API RBAC table for observability

API	Admin	Edit	View
multiclusterobservabilities.observability.open-cluster-management.io	create, read, update, and delete	-	-

To continue to learn more about securing your cluster, see Security.

1.2. CREDENTIALS

You can rotate your credentials for your Red Hat Advanced Cluster Management for Kubernetes clusters when your cloud provider access credentials have changed. Continue reading for the procedure to manually propagate your updated cloud provider credentials.

Required access: Cluster administrator

1.2.1. Provider credentials

Connection secrets for a cloud provider can be rotated. See the following list of provider credentials:

1.2.1.1. Amazon Web Services

- **aws_access_key_id**: Your provisioned cluster access key.
- **aws_secret_access_key**: Your provisioned secret access key.
  1. View the resources in the namespace that has the same name as the cluster with the expired credential.
  2. Find the secret name **<cluster_name>-<cloud_provider>-creds**. For example: **my_cluster-aws-creds1**.
  3. Edit the secret to replace the existing value with the updated value.

1.2.2. Agents

Agents are responsible for connections. See how you can rotate the following credentials:

- **registration-agent**: Connects the registration agent to the hub cluster.
- **work-agent**: Connects the work agent to the hub cluster.
  To rotate credentials, delete the **hub-kubeconfig** secret to restart the registration pods.
- **APIServer**: Connects agents and add-ons to the hub cluster.
  1. On the hub cluster, display the import command by entering the following command:

     ```
     oc get secret -n ${CLUSTER_NAME} ${CLUSTER_NAME}-import -
     ojsonpath='{.data.import\.yaml}' | base64 --decode  > import.yaml
     ```

  2. On the managed cluster, apply the **import.yaml** file. Run the following command: **oc apply -f import.yaml**.

1.3. CERTIFICATES

Various certificates are created and used throughout Red Hat Advanced Cluster Management for Kubernetes.

You can bring your own certificates. You must create a Kubernetes TLS Secret for your certificate. After you create your certificates, you can replace certain certificates that are created by the Red Hat Advanced Cluster Management installer.

Required access: Cluster administrator or team administrator.

Note: Replacing certificates is supported only on native Red Hat Advanced Cluster Management installations.

All certificates required by services that run on Red Hat Advanced Cluster Management are created during the installation of Red Hat Advanced Cluster Management. Certificates are created and managed by the Red Hat Advanced Cluster Management Certificate manager (**cert-manager**) service. The Red Hat Advanced Cluster Management Root Certificate Authority (CA) certificate is stored within the Kubernetes Secret **multicloud-ca-cert** in the hub cluster namespace. The certificate can be imported into your client truststores to access Red Hat Advanced Cluster Management Platform APIs.

See the following topics to replace certificates:

- Replacing the root CA certificate
- Replacing the management ingress certificates

1.3.1. List managed certificates

You can view a list of managed certificates that use **cert-manager** internally by running the following command:

```
oc get certificates.certmanager.k8s.io -n open-cluster-management
```

Note: If observability is enabled, there are additional namespaces where certificates are created.

1.3.2. Refresh a managed certificate

You can refresh a managed certificate by running the command in the List managed certificates section. When you identify the certificate that you need to refresh, delete the secret that is associated with the certificate. For example, you can delete a secret by running the following command:

```
oc delete secret grc-0c925-grc-secrets -n open-cluster-management
```

1.3.3. Refresh managed certificates for Red Hat Advanced Cluster Management for Kubernetes

You can refresh all managed certificates that are issued by the Red Hat Advanced Cluster Management CA. During the refresh, the Kubernetes secret that is associated with each **cert-manager** certificate is deleted. The service restarts automatically to use the certificate. Run the following command:

```
oc delete secret -n open-cluster-management $(oc get certificates.certmanager.k8s.io -n open-
cluster-management -o wide | grep multicloud-ca-issuer | awk '{print $3}')
```

The Red Hat OpenShift Container Platform certificate is not included in the Red Hat Advanced Cluster Management for Kubernetes management ingress. For more information, see the Security known issues.

1.3.4. Refresh internal certificates

You can refresh internal certificates, which are certificates that are used by Red Hat Advanced Cluster Management webhooks and the proxy server.

Complete the following steps to refresh internal certificates:

1. Delete the secret that is associated with the internal certificate by running the following command:

   ```
   oc delete secret -n open-cluster-management ocm-webhook-secret
   ```

   Note: Some services might not have a secret that needs to be deleted.

2. Restart the services that are associated with the internal certificate(s) by running the following command:

   ```
   oc delete po -n open-cluster-management ocm-webhook-679444669c-5cg76
   ```

Remember: There are replicas of many services; each service must be restarted.

View the following table for a summarized list of the pods that contain certificates and whether a secret needs to be deleted prior to restarting the pod:

Table 1.9. Pods that contain internal certificates

Service name	Namespace	Sample pod name	Secret name (if applicable)
channels-apps-open-cluster-management-webhook-svc	open-cluster-management	multicluster-operators-application-8c446664c-5lbfk	-
multicluster-operators-application-svc	open-cluster-management	multicluster-operators-application-8c446664c-5lbfk	-
multiclusterhub-operator-webhook	open-cluster-management	multiclusterhub-operator-bfd948595-mnhjc	-
ocm-webhook	open-cluster-management	ocm-webhook-679444669c-5cg76	ocm-webhook-secret
cluster-manager-registration-webhook	open-cluster-management-hub	cluster-manager-registration-webhook-fb7b99c-d8wfc	registration-webhook-serving-cert
cluster-manager-work-webhook	open-cluster-management-hub	cluster-manager-work-webhook-89b8d7fc-f4pv8	work-webhook-serving-cert

1.3.4.1. 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 po -n openshift-gatekeeper-system -l control-plane=controller-manager
```

The gatekeeper webhook certificate is rotated.

1.3.4.2. Rotating the integrity shield webhook certificate (Technology preview)

Complete the following steps to rotate the integrity shield webhook certificate:

1. Edit the IntegrityShield custom resource and add the **integrity-shield-operator-system** namespace to the excluded list of namespaces in the **inScopeNamespaceSelector** setting. Run the following command to edit the resource:

   ```
   oc edit integrityshield integrity-shield-server -n integrity-shield-operator-system
   ```

2. Delete the secret that contains the integrity shield certificate by running the following command:

   ```
   oc delete secret -n integrity-shield-operator-system ishield-server-tls
   ```

3. Delete the operator so that the secret is recreated. Be sure that the operator pod name matches the pod name on your system. Run the following command:

   ```
   oc delete po -n integrity-shield-operator-system integrity-shield-operator-controller-manager-
   64549569f8-v4pz6
   ```

4. Delete the integrity shield server pod to begin using the new certificate with the following command:

   ```
   oc delete po -n integrity-shield-operator-system integrity-shield-server-5fbdfbbbd4-bbfbz
   ```

1.3.4.3. Observability certificates

When Red Hat Advanced Cluster Management is installed there are additional namespaces where certificates are managed. The **open-cluster-management-observability** namespace and the managed cluster namespaces contain certificates managed by **cert-manager** for the observability service.

Observability certificates are automatically refreshed upon expiration. View the following list to understand the effects when certificates are automatically renewed:

- Components on your hub cluster automatically restart to retrieve the refreshed certificate.
- Red Hat Advanced Cluster Management sends the refreshed certificates to managed clusters.
- The **metrics-collector** restarts to mount the renewed certificates.
  Note: metrics-collector can push metrics to the hub cluster before and after certificates are removed. For more information about refreshing certificates across your clusters, see the Refresh internal certificates section. Be sure to specify the appropriate namespace when you refresh a certificate.

1.3.4.4. Channel certificates

CA certificates can be associated with Git channel that are a part of the Red Hat Advanced Cluster Management application management. See Using custom CA certificates for a secure HTTPS connection for more details.

Helm channels allow you to disable certificate validation. Helm channels where certificate validation is disabled, must be configured in development environments. Disabling certificate validation introduces security risks.

1.3.4.5. Managed cluster certificates

Certificates are used to authenticate managed clusters with the hub. Therefore, it is important to be aware of troubleshooting scenarios associated with these certificates. View Troubleshooting imported clusters offline after certificate change for more details.

The managed cluster certificates are refreshed automatically.

Use the certificate policy controller to create and manage certificate policies on managed clusters. See Policy controllers to learn more about controllers. Return to the Security page for more information.

1.3.5. Replacing the root CA certificate

You can replace the root CA certificate.

1.3.5.1. Prerequisites for root CA certificate

Verify that your Red Hat Advanced Cluster Management for Kubernetes cluster is running.

Back up the existing Red Hat Advanced Cluster Management for Kubernetes certificate resource by running the following command:

```
oc get cert multicloud-ca-cert -n open-cluster-management -o yaml > multicloud-ca-cert-backup.yaml
```

1.3.5.2. Creating the root CA certificate with OpenSSL

Complete the following steps to create a root CA certificate with OpenSSL:

1. Generate your certificate authority (CA) RSA private key by running the following command:

   ```
   openssl genrsa -out ca.key 4096
   ```

2. Generate a self-signed CA certificate by using your CA key. Run the following command:

   ```
   openssl req -x509 -new -nodes -key ca.key -days 400 -out ca.crt -config req.cnf
   ```

   Your **req.cnf** file might resemble the following file:

   ```
   [ req ]             # Main settings
   default_bits = 4096       # Default key size in bits.
   prompt = no               # Disables prompting for certificate values so the configuration file
   values are used.
   default_md = sha256       # Specifies the digest algorithm.
   distinguished_name = dn   # Specifies the section that includes the distinguished name
   information.
   x509_extensions = v3_ca   # The extentions to add to the self signed cert

   [ dn ]              # Distinguished name settings
   C = US                    # Country
   ST = North Carolina          # State or province
   L = Raleigh               # Locality
   O = Red Hat Open Shift    # Organization
   OU = Red Hat Advanced Container Management      # Organizational unit
   CN = www.redhat.com  # Common name.
   ```

```
[ v3_ca ]          # x509v3 extensions
basicConstraints=critical,CA:TRUE # Indicates whether the certificate is a CA certificate
during the certificate chain verification process.
```

1.3.5.3. Replacing root CA certificates

1. Create a new secret with the CA certificate by running the following command:

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

2. Edit the CA issuer to point to the BYO certificate. Run the following commnad:

```
oc edit issuer -n open-cluster-management multicloud-ca-issuer
```

3. Replace the string **mulicloud-ca-cert** with **byo-ca-cert**. Save your deployment and quit the editor.

4. Edit the management ingress deployment to reference the Bring Your Own (BYO) CA certificate. Run the following command:

```
oc edit deployment management-ingress-435ab
```

5. Replace the **multicloud-ca-cert** string with the **byo-ca-cert**. Save your deployment and quit the editor.

6. Validate the custom CA is in use by logging in to the console and view the details of the certificate being used.

1.3.5.4. Refreshing cert-manager certificates

After the root CA is replaced, all certificates that are signed by the root CA must be refreshed and the services that use those certificates must be restarted. Cert-manager creates the default Issuer from the root CA so all of the certificates issued by **cert-manager**, and signed by the default ClusterIssuer must also be refreshed.

Delete the Kubernetes secrets associated with each **cert-manager** certificate to refresh the certificate and restart the services that use the certificate. Run the following command:

```
oc delete secret -n open-cluster-management $(oc get cert -n open-cluster-management -o wide |
grep multicloud-ca-issuer | awk '{print $3}')
```

1.3.5.5. Restoring root CA certificates

To restore the root CA certificate, update the CA issuer by completing the following steps:

1. Edit the CA issuer. Run the following command:

```
oc edit issuer -n open-cluster-management multicloud-ca-issuer
```

2. Replace the **byo-ca-cert** string with **multicloud-ca-cert** in the editor. Save the issuer and quit the editor.

3. Edit the management ingress depolyment to reference the original CA certificate. Run the following command:

   ```
   oc edit deployment management-ingress-435ab
   ```

4. Replace the **byo-ca-cert** string with the **multicloud-ca-cert** string. Save your deployment and quit the editor.

5. Delete the BYO CA certificate. Run the following commnad:

   ```
   oc delete secret -n open-cluster-management byo-ca-cert
   ```

Refresh all **cert-manager** certificates that use the CA. For more information, see the forementioned section, Refreshing *cert-manager* certificates.

See Certificates for more information about certificates that are created and managed by Red Hat Advanced Cluster Management for Kubernates.

1.3.6. Replacing the management ingress certificates

You can replace management ingress certificates.

1.3.6.1. Prerequisites to replace management ingress certificate

Prepare and have your **management-ingress** certificates and private keys ready. If needed, you can generate a TLS certificate by using OpenSSL. Set the common name parameter,**CN**, on the certificate to **manangement-ingress**. If you are generating the certificate, include the following settings:

- Include the route name for Red Hat Advanced Cluster Management for Kubernetes as the domain name in your certificate Subject Alternative Name (SAN) list.
  - The service name for the management ingress: **management-ingress**.
  - Include the route name for Red Hat Advanced Cluster Management for Kubernetes. Receive the route name by running the following command:

    ```
    oc get route -n open-cluster-management
    ```

    You might receive the following response:

    ```
    multicloud-console.apps.grchub2.dev08.red-chesterfield.com
    ```

  - Add the localhost IP address: **127.0.0.1**.
  - Add the localhost entry: **localhost**.

1.3.6.1.1. Example configuration file for generating a certificate

The following example configuration file and OpenSSL commands provide an example for how to generate a TLS certificate by using OpenSSL. View the following **csr.cnf** configuration file, which defines the configuration settings for generating certificates with OpenSSL.

```
[ req ]              # Main settings
default_bits = 2048       # Default key size in bits.
```

```
prompt = no              # Disables prompting for certificate values so the configuration file values are
used.
default_md = sha256       # Specifies the digest algorithm.
req_extensions = req_ext  # Specifies the configuration file section that includes any extensions.
distinguished_name = dn   # Specifies the section that includes the distinguished name information.

[ dn ]              # Distinguished name settings
C = US                   # Country
ST = North Carolina             # State or province
L = Raleigh                # Locality
O = Red Hat Open Shift     # Organization
OU = Red Hat Advanced Container Management        # Organizational unit
CN = management-ingress  # Common name.

[ req_ext ]          # Extensions
subjectAltName = @alt_names # Subject alternative names

[ alt_names ]        # Subject alternative names
DNS.1 = management-ingress
DNS.2 = multicloud-console.apps.grchub2.dev08.red-chesterfield.com
DNS.3 = localhost
DNS.4 = 127.0.0.1

[ v3_ext ]          # x509v3 extensions
authorityKeyIdentifier=keyid,issuer:always  # Specifies the public key that corresponds to the private
key that is used to sign a certificate.
basicConstraints=CA:FALSE                   # Indicates whether the certificate is a CA certificate during
the certificate chain verification process.
#keyUsage=keyEncipherment,dataEncipherment   # Defines the purpose of the key that is contained
in the certificate.
extendedKeyUsage=serverAuth                  # Defines the purposes for which the public key can be
used.
subjectAltName=@alt_names                   # Identifies the subject alternative names for the identify
that is bound to the public key by the CA.
```

Note: Be sure to update the SAN labeled, **DNS.2** with the correct hostname for your management ingress.

1.3.6.1.2. OpenSSL commands for generating a certificate

The following OpenSSL commands are used with the preceding configuration file to generate the required TLS certificate.

1. Generate your certificate authority (CA) RSA private key:

   ```
   openssl genrsa -out ca.key 4096
   ```

2. Generate a self-signed CA certificate by using your CA key:

   ```
   openssl req -x509 -new -nodes -key ca.key -subj "/C=US/ST=North
   Carolina/L=Raleigh/O=Red Hat OpenShift" -days 400 -out ca.crt
   ```

3. Generate the RSA private key for your certificate:

   ```
   openssl genrsa -out ingress.key 4096
   ```

4. Generate the Certificate Signing request (CSR) by using the private key:

   ```
   openssl req -new -key ingress.key -out ingress.csr -config csr.cnf
   ```

5. Generate a signed certificate by using your CA certificate and key and CSR:

   ```
   openssl x509 -req -in ingress.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out ingress.crt -
   sha256 -days 300 -extensions v3_ext -extfile csr.cnf
   ```

6. Examine the certificate contents:

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

1.3.6.2. Replace the Bring Your Own (BYO) ingress certificate

Complete the following steps to replace your BYO ingress certificate:

1. Create the **byo-ingress-tls** secret by using your certificate and private key. Run the following command:

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

2. Verify that the secret is created in the correct namespace with the following command:

   ```
   kubectl get secret -n open-cluster-management | grep -e byo-ingress-tls-secret -e byo-ca-cert
   ```

3. Create a secret containing the CA certificate by running the following command:

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

4. Edit the management ingress deployment and get the name of the deployment with the following commands:

   ```
   export MANAGEMENT_INGRESS=`oc get deployment -o custom-columns=:.metadata.name
   | grep management-ingress`

   oc edit deployment $MANAGEMENT_INGRESS -n open-cluster-management
   ```

   - Replace the **multicloud-ca-cert** string with **byo-ca-cert**.
   - Replace the **$MANAGEMENT_INGRESS-tls-secret** string with **byo-ingress-tls-secret**.
   - Save your deployment and close the editor. + The management ingress automatically restarts.

5. Verify that the current certificate is your certificate, and that all console access and login functionality remain the same.

1.3.6.3. Restore the default self-signed certificate for management ingress

1. Edit the management ingress deployment. Replace the string **multicloud-ca-cert** with **byo-ca-cert** and get the name of the deployment with the following commands:

```
export MANAGEMENT_INGRESS=`oc get deployment -o custom-columns=:.metadata.name
| grep management-ingress`

oc edit deployment $MANAGEMENT_INGRESS -n open-cluster-management
```

a. Replace the **byo-ca-cert** string with **multicloud-ca-cert**.

b. Replace the **byo-ingress-tls-secret** string with the **$MANAGEMENT_INGRESS-tls-secret**.

c. Save your deployment and close the editor. The management ingress automatically restarts.

2. After all pods are restarted, navigate to the Red Hat Advanced Cluster Management for Kubernetes console from your browser.

3. Verify that the current certificate is your certificate, and that all console access and login functionality remain the same.

4. Delete the Bring Your Own (BYO) ingress secret and ingress CA certificate by running the following commands:

```
oc delete secret -n open-cluster-management byo-ingress-tls-secret
oc delete secret -n open-cluster-management byo-ca-cert
```

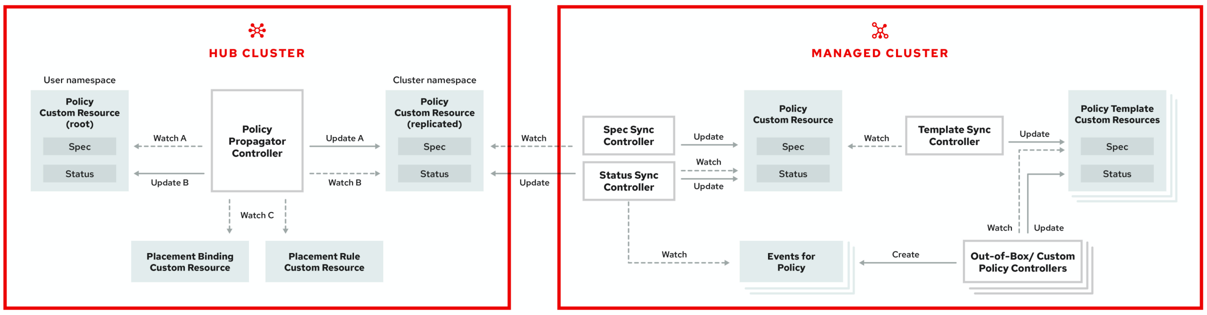
See Certificates for more information about certificates that are created and managed by Red Hat Advanced Cluster Management. Return to the Security page for more information on securing your cluster.

CHAPTER 2. GOVERNANCE AND RISK

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.

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 defined policies, processes, and procedures to manage security and compliance from a central interface page. View the following diagram of the governance architecture:



The governance architecture is composed of the following components:

- **Governance and risk dashboard**: Provides a summary of your cloud governance and risk details, which include policy and cluster violations.
  Notes:
  - When a policy is propagated to a managed cluster, the replicated policy is named **namespaceName.policyName**. When you create a policy, make sure that the length of the **namespaceName.policyName** must not exceed 63 characters due to the Kubernetes limit for object names.
  - When you search for a policy in the hub cluster, you might also receive the name of the replicated policy on your managed cluster. For example, if you search for **policy-dhaz-cert**, the following policy name from the hub cluster might appear: **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. See the **policy-collection** repository to view examples of the predefined policies, and instructions on deploying policies to your cluster. You can also contribute custom policy controllers and policies.
- **Policy controller**: Evaluates one or more policies on the managed cluster against your specified control and generates Kubernetes events for violations. Violations are propagated to the hub cluster. Policy controllers that are included in your installation are the following: Kubernetes configuration, Certificate, and IAM. You can also create a custom policy controller.
- **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. Learn how to

contribute and deploy policies using GitOps. For more information, see Deploy policies using GitOps. Learn how to integrate third-party policies with Red Hat Advanced Cluster Management for Kubernetes. For more information, see Integrate third-party policy controllers.

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 and risk dashboard.

- Policy overview
- Policy controllers
- Supported policies
- Manage security policies

2.2. POLICY OVERVIEW

Use the Red Hat Advanced Cluster Management for Kubernetes security policy framework to create custom policy controllers and other policies. Kubernetes custom resource definition (CRD) instance are used to create policies. For more information about CRDs, see Extend the Kubernetes API with CustomResourceDefinitions.

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

The policy requires a *PlacementRule* that defines the clusters that the policy document is applied to, and a *PlacementBinding* that binds the Red Hat Advanced Cluster Management for Kubernetes policy to the placement rule.

Important:

- You must create a **PlacementRule** to apply your policies to the managed cluster, and bind the **PlacementRule** with a **PlacementBinding**.
- 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.

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 the explained parameter fields:

```
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:
spec:
  policy-templates:
    - objectDefinition:
        apiVersion:
        kind:
        metadata:
          name:
        spec:
  remediationAction:
  disabled:

---
apiVersion: apps.open-cluster-management.io/v1
kind: PlacementBinding
metadata:
  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

Field	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	Description
metadata.annotations	Optional. Used to specify a set of security details that describes the set of standards the policy is trying to validate. All annotations documented here are represented as a string that contains a comma-separated list. **Note**: You can view policy violations based on the standards and categories that you define for your policy on the *Policies* page, from the console.
annotations.policy.open-cluster-management.io/standards	The name or names of security standards the policy is related to. For example, National Institute of Standards and Technology (NIST) and Payment Card Industry (PCI).
annotations.policy.open-cluster-management.io/categories	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	The name of the security control that is being checked. For example, the certificate policy controller.
spec.policy-templates	Required. Used to create one or more policies to apply to a managed cluster.
spec.disabled	Required. Set the value to **true** or **false**. The **disabled** parameter provides the ability to enable and disable your policies.
spec.remediationAction	Optional. Specifies the remediation of your policy. The parameter values are **enforce** and **inform**. If specified, the **spec.remediationAction** value that is defined overrides the **remediationAction** parameter defined in the child policy, from the **policy-templates** section. For example, if **spec.remediationAction** value section is set to **enforce**, then the **remediationAction** in the **policy-templates** section is set to **enforce** during runtime. **Important**: Some policies might not support the enforce feature.

2.2.3. Policy sample file

```
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
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:
            exclude: ["kube-*"]
            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"]}
```

See Managing security policies to create and update a policy. You can also enable and updateRed Hat Advanced Cluster Management policy controllers to validate the compliance of your policies. Refer to Policy controllers. To learn more policy topics, see Governance and risk.

2.3. POLICY CONTROLLERS

Policy controllers monitor and report whether your cluster is compliant with a policy. Use the Red Hat Advanced Cluster Management for Kubernetes policy framework by using the out of the box policy templates to apply predefined policy controllers, and policies. The policy controllers are Kubernetes custom resource definition (CRD) instance. For more information about CRDs, see Extend the Kubernetes API with CustomResourceDefinitions. Policy controllers remediate policy violations to make the cluster status be compliant.

You can create custom policies and policy controllers with the product policy framework. See Creating a custom policy controller for more information.

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

Refer to Governance and risk for more topics about managing your policies.

2.3.1. Kubernetes configuration policy controller

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

The configuration policy controller communicates with the local Kubernetes API server to get the list of your configurations that are in your cluster. For more information about CRDs, see Extend the Kubernetes API with CustomResourceDefinitions.

The configuration policy controller is created on the hub cluster during installation. Configuration policy controller supports the **enforce** feature and monitors the compliance of the following policies:

- Memory usage policy
- Namespace policy
- Image vulnerability policy
- Pod policy
- Pod security policy

- Role policy
- Role binding policy
- Security content constraints (SCC) policy
- ETCD encryption policy
- Compliance operator policy

When the **remediationAction** for the configuration policy is set to **enforce**, the controller creates a replicate policy on the target managed clusters.

2.3.1.1. Configuration policy controller YAML structure

```
Name:         configuration-policy-example
Namespace:
Labels:
APIVersion:   policy.open-cluster-management.io/v1
Kind:         ConfigPolicy
Metadata:
  Finalizers:
    finalizer.policy.open-cluster-management.io
Spec:
  Conditions:
    Ownership:
    NamespaceSelector:
      Exclude:
      Include:
    RemediationAction:
 Status:
   CompliancyDetails:
     Configuration-Policy-Example:
       Default:
       Kube - Public:
   Compliant:          Compliant
 Events:
```

2.3.1.2. Configuration policy sample

```
apiVersion: policy.open-cluster-management.io/v1
kind: ConfigPolicy
metadata:
  name: policy-config
spec:
  namespaceSelector:
    include: ["default"]
    exclude: []
  remediationAction: inform
    severity: low
    object-templates:
    - complianceType: musthave
      objectDefinition:
        apiVersion: v1
        kind: Pod
```

```
metadata:
  name: nginx-pod
spec:
  containers:
  - image: nginx:1.7.9
    name: nginx
    ports:
   - containerPort: 80
```

2.3.1.3. Configuration policy YAML table

Table 2.1. Parameter table

Field	Description
apiVersion	Required. Set the value to **policy.open-cluster-management.io/v1**.
kind	Required. Set the value to **ConfigPolicy** to indicate the type of policy.
metadata.name	Required. The name of the policy.
spec	Required. Specifications of which configuration policy to monitor and how to remediate them.
spec.namespace	Required for namespaced objects or resources. The namespaces in the hub cluster that the policy is applied to. Enter at least one namespace for the **include** parameter, which are the namespaces you want to apply to the policy to. The **exclude** parameter specifies the namespaces you explicitly do not want to apply the policy to.
spec.remediationAction	Required. Specifies the remediation of your policy. Enter **inform**
spec.remediationAction.severity	Required. Specifies the severity when the policy is non-compliant. Use the following parameter values: **low**, **medium**, or **high**.

Field	Description
spec.remediationAction.complianceType	Required. Used to list expected behavior for roles and other Kubernetes object that must be evaluated or applied to the managed clusters. You must use the following verbs as parameter values: **mustonlyhave**: Indicates that an object must exist with the exact name and relevant fields. **musthave**: Indicates an object must exist with the same name as specified object-template. The other fields in the template are a subset of what exists in the object. **mustnothave**: Indicated that an object with the same name or labels cannot exist and need to be deleted, regardless of the specification or rules.

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.

Learn how to create and customize policies, see Manage security policies . Refer to Policy controllers for more details about controllers.

2.3.2. Certificate policy controller

Certificate policy controller can be used to detect certificates that are close to expiring, and detect time durations (hours) that are too long or contain DNS names that fail to match specified patterns.

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. For more information about CRDs, see Extend the Kubernetes API with CustomResourceDefinitions.

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

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
  namespace:
  labels: category=system-and-information-integrity
spec:
  namespaceSelector:
    include: ["default"]
    exclude: ["kube-*"]
  remediationAction:
  severity:
  minimumDuration:
  minimumCADuration:
  maximumDuration:
  maximumCADuration:
  allowedSANPattern:
  disallowedSANPattern:
```

2.3.2.1.1. Certificate policy controller YAML table

Table 2.2. Parameter table

Field	Description
apiVersion	Required. Set the value to **policy.open-cluster-management.io/v1**.
kind	Required. Set the value to **CertificatePolicy** to indicate the type of policy.
metadata.name	Required. The name to identify the policy.
metadata.namespace	Required. The namespaces within the managed cluster where the policy is created.

Field	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	Required. Specifications of which certificates to monitor and refresh.
spec.namespaceSelector	Required. Managed cluster namespace to which you want to apply the policy. Enter parameter values for **Include** and **Exclude**. **Notes**: • When you create multiple certificate policies and apply them to the same managed cluster, each policy **namespaceSelector** must be assigned a different value. • If the **namespaceSelector** for the certificate policy controller does not match any namespace, the policy is considered compliant.
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**, or **high**.
spec.minimumDuration	Required. When a value is not specified, the default value is **100h**. This parameter specifies the smallest duration (in hours) before a certificate is considered non-compliant. The parameter value uses the time duration format from Golang. See Golang Parse Duration for more information.
spec.minimumCADuration	Optional. Set a value to identify signing certificates that might expire soon with a different value from other certificates. If the parameter value is not specified, the CA certificate expiration is the value used for the **minimumDuration**. See Golang Parse Duration for more information.
spec.maximumDuration	Optional. Set a value to identify certificates that have been created with a duration that exceeds your desired limit. The parameter uses the time duration format from Golang. See Golang Parse Duration for more information.

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

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.

Learn how to manage a certificate policy, see Managing certificate policies for more details. Refer to Policy controllers for more topics.

2.3.3. IAM policy controller

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 controller checks for compliance of the number of cluster administrators that you allow in your cluster. IAM policy controller communicates with the local Kubernetes API server. For more information, see Extend the Kubernetes API with CustomResourceDefinitions.

The IAM policy controller runs on your managed cluster.

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:
  severity:
  namespaceSelector:
    include:
    exclude:
  remediationAction:
  maxClusterRoleBindingUsers:
```

2.3.3.2. IAM policy YAMl table

View the following parameter table for descriptions:

Table 2.3. Parameter table

Field	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	Required. Add configuration details for your policy.
spec.severity	Optional. Informs the user of the severity when the policy is non-compliant. Use the following parameter values: **low**, **medium**, or **high**.
spec.namespaceSelector	Required. The namespaces within the hub cluster that the policy is applied to. Enter at least one namespace for the **include** parameter, which are the namespaces you want to apply to the policy to. The **exclude** parameter specifies the namespaces you explicitly do not want to apply the policy to. **Note**: A namespace that is specified in the object template of a policy controller overrides the namespace in the preceding parameter values.
spec.remediationAction	Optional. Specifies the remediation of your policy. Enter **inform**.
spec.maxClusterRoleBindingUsers	Required. Maximum number of IAM role bindings that are available before a policy is considered non-compliant.

2.3.3.3. IAM policy sample

See **policy-limitclusteradmin.yaml** to view the IAM policy sample. Learn how to manage an IAM policy, see Managing IAM policies for more details. Refer to Policy controllers for more topics.

2.3.4. 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

2.3.5. Creating a custom policy controller

Learn to write, apply, view, and update your custom policy controllers. You can create a YAML file for your policy controller to deploy onto your cluster. View the following sections to create a policy controller:

2.3.5.1. Writing a policy controller

Use the policy controller framework that is in the **governance-policy-framework** repository. Complete the following steps to create a policy controller:

1. Clone the **governance-policy-framework** repository by running the following command:

```
git clone git@github.com:stolostron/governance-policy-framework.git
```

2. Customize the controller policy by updating the policy schema definition. Your policy might resemble the following content:

```
metadata:
  name: samplepolicies.policies.open-cluster-management.io
spec:
  group: policy.open-cluster-management.io
  names:
    kind: SamplePolicy
    listKind: SamplePolicyList
    plural: samplepolicies
    singular: samplepolicy
```

3. Update the policy controller to watch for the **SamplePolicy** kind. Run the following command:

```
for file in $(find . -name "*.go" -type f); do  sed -i "" "s/SamplePolicy/g" $file; done
for file in $(find . -name "*.go" -type f); do  sed -i "" "s/samplepolicy-controller/samplepolicy-
controller/g" $file; done
```

4. Recompile and run the policy controller by completing the following steps:

   a. Log in to your cluster.

   b. Select the user icon, then click **Configure client**.

   c. Copy and paste the configuration information into your command line, and press **Enter**.

d. Run the following commands to apply your policy CRD and start the controller:

```
export GO111MODULE=on

kubectl apply -f deploy/crds/policy.open-cluster-management.io_samplepolicies_crd.yaml

export WATCH_NAMESPACE=<cluster_namespace_on_hub>

go run cmd/manager/main.go
```

You might receive the following output that indicates that your controller runs:

```
{“level”:”info”,”ts”:1578503280.511274,”logger”:”controller-
runtime.manager”,”msg”:”starting metrics server”,”path”:”/metrics”}
{“level”:”info”,”ts”:1578503281.215883,”logger”:”controller-
runtime.controller”,”msg”:”Starting Controller”,”controller”:”samplepolicy-controller”}
{“level”:”info”,”ts”:1578503281.3203468,”logger”:”controller-
runtime.controller”,”msg”:”Starting workers”,”controller”:”samplepolicy-controller”,”worker
count”:1}
Waiting for policies to be available for processing…
```

e. Create a policy and verify that the controller retrieves it and applies the policy onto your cluster. Run the following command:

```
kubectl apply -f deploy/crds/policy.open-cluster-management.io_samplepolicies_crd.yaml
```

When the policy is applied, a message appears to indicate that policy is monitored and detected by your custom controller. The mesasge might resemble the following contents:

```
{"level":"info","ts":1578503685.643426,"logger":"controller_samplepolicy","msg":"Reconciling
SamplePolicy","Request.Namespace":"default","Request.Name":"example-samplepolicy"}
{"level":"info","ts":1578503685.855259,"logger":"controller_samplepolicy","msg":"Reconciling
SamplePolicy","Request.Namespace":"default","Request.Name":"example-samplepolicy"}
Available policies in namespaces:
namespace = kube-public; policy = example-samplepolicy
namespace = default; policy = example-samplepolicy
namespace = kube-node-lease; policy = example-samplepolicy
```

5. Check the **status** field for compliance details by running the following command:

```
kubectl describe SamplePolicy example-samplepolicy -n default
```

Your output might resemble the following contents:

```
status:
  compliancyDetails:
    example-samplepolicy:
      cluster-wide:
      - 5 violations detected in namespace `cluster-wide`, there are 0 users violations
        and 5 groups violations
      default:
      - 0 violations detected in namespace `default`, there are 0 users violations
        and 0 groups violations
      kube-node-lease:
```

```
    - 0 violations detected in namespace `kube-node-lease`, there are 0 users violations
      and 0 groups violations
    kube-public:
    - 1 violations detected in namespace `kube-public`, there are 0 users violations
      and 1 groups violations
  compliant: NonCompliant
```

6. Change the policy rules and policy logic to introduce new rules for your policy controller. Complete the following steps:

   a. Add new fields in your YAML file by updating the **SamplePolicySpec**. Your specification might resemble the following content:

   ```
   spec:
     description: SamplePolicySpec defines the desired state of SamplePolicy
     properties:
       labelSelector:
         additionalProperties:
           type: string
         type: object
       maxClusterRoleBindingGroups:
         type: integer
       maxClusterRoleBindingUsers:
         type: integer
       maxRoleBindingGroupsPerNamespace:
         type: integer
       maxRoleBindingUsersPerNamespace:
         type: integer
   ```

   b. Update the **SamplePolicySpec** structure in the samplepolicy_controller.go with new fields.

   c. Update the **PeriodicallyExecSamplePolicies** function in the **samplepolicy_controller.go** file with new logic to run the policy controller. View an example of the **PeriodicallyExecSamplePolicies** field, see stolostron/multicloud-operators-policy-controller.

   d. Recompile and run the policy controller. See Writing a policy controller

Your policy controller is functional.

2.3.5.2. Deploying your controller to the cluster

Deploy your custom policy controller to your cluster and integrate the policy controller with the Governance and risk dashboard. Complete the following steps:

1. Build the policy controller image by running the following command:

   ```
   make build
   docker build . -f build/Dockerfile -t <username>/multicloud-operators-policy-controller:latest
   ```

2. Run the following command to push the image to a repository of your choice. For example, run the following commands to push the image to Docker Hub:

```
docker login

docker push <username>/multicloud-operators-policy-controller
```

3. Configure **kubectl** to point to a cluster managed by Red Hat Advanced Cluster Management for Kubernetes.

4. Replace the operator manifest to use the built-in image name and update the namespace to watch for policies. The namespace must be the cluster namespace. Your manifest might resemble the following contents:

```
sed -i "" 's|open-cluster-management/multicloud-operators-policy-controller|ycao/multicloud-operators-policy-controller|g' deploy/operator.yaml
sed -i "" 's|value: default|value: <namespace>|g' deploy/operator.yaml
```

5. Update the RBAC role by running the following commands:

```
sed -i "" 's|samplepolicies|testpolicies|g' deploy/cluster_role.yaml
sed -i "" 's|namespace: default|namespace: <namespace>|g'
deploy/cluster_role_binding.yaml
```

6. Deploy your policy controller to your cluster:

   a. Set up a service account for cluster by runnng the following command:

   ```
   kubectl apply -f deploy/service_account.yaml -n <namespace>
   ```

   b. Set up RBAC for the operator by running the following commands:

   ```
   kubectl apply -f deploy/role.yaml -n <namespace>

   kubectl apply -f deploy/role_binding.yaml -n <namespace>
   ```

   c. Set up RBAC for your policy controller. Run the following commands:

   ```
   kubectl apply -f deploy/cluster_role.yaml
   kubectl apply -f deploy/cluster_role_binding.yaml
   ```

   d. Set up a custom resource definition (CRD) by running the following command:

   ```
   kubectl apply -f deploy/crds/policies.open-cluster-management.io_samplepolicies_crd.yaml
   ```

   e. Deploy the **multicloud-operator-policy-controller** by running the following command:

   ```
   kubectl apply -f deploy/operator.yaml -n <namespace>
   ```

   f. Verify that the controller is functional by running the following command:

   ```
   kubectl get pod -n <namespace>
   ```

7. You must integrate your policy controller by creating a **policy-template** for the controller to monitor. For more information, see Creating a cluster security policy from the console .

2.3.5.2.1. Scaling your controller deployment

Policy controller deployments do not support deletetion or removal. You can scale your deployment to update which pods the deployment is applied to. Complete the following steps:

1. Log in to your managed cluster.
2. Navigate to the deployment for your custom policy controller.
3. Scale the deployment. When you scale your deployment to zero pods, the policy controler deployment is disabled.

For more information on deployments, see OpenShift Container Platform Deployments.

Your policy controller is deployed and integrated on your cluster. View the product policy controllers, see Policy controllers for more information.

2.4. 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.

Note: You can copy and paste an existing policy in to the *Policy YAML*. The values for the parameter fields are automatically entered when you paste your existing policy. You can also search the contents in your policy YAML file with the search feature.

View the following policy samples to view how specific policies are applied:

- Image vulnerability policy
- Memory usage policy
- Namespace policy
- Pod nginx policy
- Pod security policy
- Role policy
- Role binding policy
- Security context constraints policy
- ETCD encryption policy
- Compliance operator policy
- E8 scan policy

Refer to Governance and risk for more topics.

2.4.1. Memory usage policy

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.

2.4.1.1. 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: policy-limitrange
  namespace:
spec:
  complianceType:
  remediationAction:
  namespaces:
    exclude:
    include:
  object-templates:
    - complianceType:
      objectDefinition:
        apiVersion:
        kind:
        metadata:
          name:
        spec:
          limits:
          - default:
              memory:
            defaultRequest:
              memory:
            type:
        ...
```

2.4.1.2. Memory usage policy table

Field	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.namespaces	Optional.

Field	Description
spec.namespace	Required. The namespaces within the hub cluster that the policy is applied to. Enter parameter values for **include**, which are the namespaces you want to apply to the policy to. The **exclude** parameter specifies the namespaces you explicitly do not want to apply the policy to. **Note**: A namespace that is specified in the object template of a policy controller overrides the namespace in the corresponding parent policy.
remediationAction	Optional. Specifies the remediation of your policy. The parameter values are **enforce** and **inform**. **Important**: Some policies might not support the enforce feature.
disabled	Required. Set the value to **true** or **false**. The **disabled** parameter provides the ability to enable and disable your policies.
spec.complianceType	Required. Set the value to **"musthave"**
spec.object-template	Optional. Used to list any other Kubernetes object that must be evaluated or applied to the managed clusters.

2.4.1.3. Memory usage policy sample

See the **policy-limitmemory.yaml** to view a sample of the policy. View Managing memory usage policies for more information. Refer to Kubernetes configuration policy controller to view other configuration policies that are monitored by the controller.

2.4.2. Namespace policy

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.

2.4.2.1. Namespace policy YAML structure

```
apiVersion: policy.open-cluster-management.io/v1
kind: Policy
metadata:
  name: policy-namespace-1
  namespace:
spec:
  complianceType:
  remediationAction:
```

```
namespaces:
  exclude:
  include:
object-templates:
  - complianceType:
    objectDefinition:
      kind:
      apiVersion:
      metadata:
        name:
  ...
```

2.4.2.2. Namespace policy YAML table

Field	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.namespaces	Optional.
spec.namespace	Required. The namespaces within the hub cluster that the policy is applied to. Enter parameter values for **include**, which are the namespaces you want to apply to the policy to. The **exclude** parameter specifies the namespaces you explicitly do not want to apply the policy to. **Note**: A namespace that is specified in the object template of a policy controller overrides the namespace in the corresponding parent policy.
remediationAction	Optional. Specifies the remediation of your policy. The parameter values are **enforce** and **inform**. **Important**: Some policies might not support the enforce feature.
disabled	Required. Set the value to **true** or **false**. The **disabled** parameter provides the ability to enable and disable your policies.
spec.complianceType	Required. Set the value to **"musthave"**
spec.object-template	Optional. Used to list any other Kubernetes object that must be evaluated or applied to the managed clusters.

2.4.2.3. Namespace policy sample

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

View Managing namespace policies for more information. Refer to Kubernetes configuration policy controller to learn about other configuration policies.

2.4.3. 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.

Note: Image vulnerability policy is not functional during a disconnected installation.

2.4.3.1. Image vulnerability policy YAML structure

```
apiVersion: policy.open-cluster-management.io/v1
kind: Policy
metadata:
  name: policy-imagemanifestvulnpolicy
  namespace: default
  annotations:
    policy.open-cluster-management.io/standards: NIST-CSF
    policy.open-cluster-management.io/categories: DE.CM Security Continuous Monitoring
    policy.open-cluster-management.io/controls: DE.CM-8 Vulnerability Scans
spec:
  remediationAction:
  disabled:
  policy-templates:
  - objectDefinition:
      apiVersion: policy.open-cluster-management.io/v1
      kind: ConfigurationPolicy
      metadata:
        name:
      spec:
        remediationAction:
        severity: high
        object-templates:
          - complianceType:
            objectDefinition:
              apiVersion: operators.coreos.com/v1alpha1
              kind: Subscription
              metadata:
                name: container-security-operator
                namespace:
              spec:
                channel:
                installPlanApproval:
                name:
                source:
                sourceNamespace:
```

```
  - objectDefinition:
      apiVersion: policy.open-cluster-management.io/v1
      kind: ConfigurationPolicy
      metadata:
        name:
      spec:
        remediationAction:
        severity:
        namespaceSelector:
          exclude:
          include:
        object-templates:
          - complianceType:
            objectDefinition:
              apiVersion: secscan.quay.redhat.com/v1alpha1
              kind: ImageManifestVuln # checking for a kind
---
apiVersion: policy.open-cluster-management.io/v1
kind: PlacementBinding
metadata:
  name: binding-policy-imagemanifestvulnpolicy
  namespace: default
placementRef:
  name:
  kind:
  apiGroup:
subjects:
- name:
  kind:
  apiGroup:
---
apiVersion: apps.open-cluster-management.io/v1
kind: PlacementRule
metadata:
  name: placement-policy-imagemanifestvulnpolicy
  namespace: default
spec:
  clusterConditions:
  - status:
    type:
  clusterSelector:
    matchExpressions:
      []  # selects all clusters if not specified
```

2.4.3.2. Image vulnerability policy YAML table

Field	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	Description
metadata.name	Required. The name for identifying the policy resource.
metadata.namespaces	Optional.
spec.namespace	Required. The namespaces within the hub cluster that the policy is applied to. Enter parameter values for **include**, which are the namespaces you want to apply to the policy to. The **exclude** parameter specifies the namespaces you explicitly do not want to apply the policy to. **Note**: A namespace that is specified in the object template of a policy controller overrides the namespace in the corresponding parent policy.
remediationAction	Optional. Specifies the remediation of your policy. The parameter values are **enforce** and **inform**. **Important**: Some policies might not support the enforce feature.
disabled	Required. Set the value to **true** or **false**. The **disabled** parameter provides the ability to enable and disable your policies.
spec.complianceType	Required. Set the value to **"musthave"**
spec.object-template	Optional. Used to list any other Kubernetes object that must be evaluated or applied to the managed clusters.

2.4.3.3. Image vulnerability policy sample

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

2.4.4. Pod policy

Kubernetes configuration policy controller monitors the status of you 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.

2.4.4.1. Pod policy YAML structure

```
apiVersion: policy.open-cluster-management.io/v1
kind: Policy
metadata:
  name: policy-pod
  namespace:
spec:
  complianceType:
  remediationAction:
  namespaces:
    exclude:
    include:
  object-templates:
    - complianceType:
      objectDefinition:
        apiVersion:
        kind: Pod # pod must exist
        metadata:
          name:
        spec:
          containers:
          - image:
            name:
            ports:
            - containerPort:
    ...
```

2.4.4.2. Pod policy table

Field	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.namespaces	Optional.
spec.namespace	Required. The namespaces within the hub cluster that the policy is applied to. Enter parameter values for **include**, which are the namespaces you want to apply to the policy to. The **exclude** parameter specifies the namespaces you explicitly do not want to apply the policy to. **Note**: A namespace that is specified in the object template of a policy controller overrides the namespace in the corresponding parent policy.
remediationAction	Optional. Specifies the remediation of your policy. The parameter values are **enforce** and **inform**. **Important**: Some policies might not support the enforce feature.

Field	Description
disabled	Required. Set the value to **true** or **false**. The **disabled** parameter provides the ability to enable and disable your policies.
spec.complianceType	Required. Set the value to **"musthave"**
spec.object-template	Optional. Used to list any other Kubernetes object that must be evaluated or applied to the managed clusters.

2.4.4.3. Pod policy sample

See **policy-pod.yaml** to view the policy sample. Learn how to manage a pod policy, see Managing pod policies for more details.

Refer to Kubernetes configuration policy controller to view other configuration policies that are monitored by the configuration controller. See Manage security policies to manage other policies.

2.4.5. Pod security policy

Kubernetes configuration policy controller monitors the status of the pod security policy. Apply a pod security policy to secure pods and containers. For more information, see *Pod Security Policies* in the Kubernetes documentation. Learn more details about the pod security policy structure in the following sections.

2.4.5.1. Pod security policy YAML structure

```
apiVersion: policy.open-cluster-management.io/v1
kind: Policy
metadata:
  name: policy-podsecuritypolicy
  namespace:
spec:
  complianceType:
  remediationAction:
  namespaces:
    exclude:
    include:
  object-templates:
    - complianceType:
      objectDefinition:
        apiVersion:
        kind: PodSecurityPolicy # no privileged pods
        metadata:
          name:
          annotations:
        spec:
          privileged:
```

```
          allowPrivilegeEscalation:
          allowedCapabilities:
          volumes:
          hostNetwork:
          hostPorts:
          hostIPC:
          hostPID:
          runAsUser:
            rule:
          seLinux:
            rule:
          supplementalGroups:
            rule:
          fsGroup:
            rule:
      ...
```

2.4.5.2. Pod security policy table

Field	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.namespaces	Optional.
spec.namespace	Required. The namespaces within the hub cluster that the policy is applied to. Enter parameter values for **include**, which are the namespaces you want to apply to the policy to. The **exclude** parameter specifies the namespaces you explicitly do not want to apply the policy to. **Note**: A namespace that is specified in the object template of a policy controller overrides the namespace in the corresponding parent policy.
remediationAction	Optional. Specifies the remediation of your policy. The parameter values are **enforce** and **inform**. **Important**: Some policies might not support the enforce feature.
disabled	Required. Set the value to **true** or **false**. The **disabled** parameter provides the ability to enable and disable your policies.

Field	Description
spec.complianceType	Required. Set the value to **"musthave"**
spec.object-template	Optional. Used to list any other Kubernetes object that must be evaluated or applied to the managed clusters.

2.4.5.3. Pod security policy sample

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

2.4.6. Role policy

Kubernetes configuration policy controller monitors the status of role policies. Define roles in the **object-template** to set rules and permissions for specific roles in your cluster. Learn more details about the role policy structure in the following sections.

2.4.6.1. Role policy YAML structure

```
apiVersion: policy.open-cluster-management.io/v1
kind: Policy
metadata:
  name: policy-role
  namespace:
  annotations:
    policy.open-cluster-management.io/standards: NIST-CSF
    policy.open-cluster-management.io/categories: PR.AC Identity Management Authentication and
Access Control
    policy.open-cluster-management.io/controls: PR.AC-4 Access Control
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 # will be overridden by remediationAction in parent policy
          severity: high
          namespaceSelector:
            exclude: ["kube-*"]
            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
  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:
      []

        ...
```

2.4.6.2. Role policy table

Field	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.namespaces	Optional.

Field	Description
spec.namespace	Required. The namespaces within the hub cluster that the policy is applied to. Enter parameter values for **include**, which are the namespaces you want to apply to the policy to. The **exclude** parameter specifies the namespaces you explicitly do not want to apply the policy to. **Note**: A namespace that is specified in the object template of a policy controller overrides the namespace in the corresponding parent policy.
remediationAction	Optional. Specifies the remediation of your policy. The parameter values are **enforce** and **inform**. **Important**: Some policies might not support the enforce feature.
disabled	Required. Set the value to **true** or **false**. The **disabled** parameter provides the ability to enable and disable your policies.
spec.complianceType	Required. Set the value to **"musthave"**
spec.object-template	Optional. Used to list any other Kubernetes object that must be evaluated or applied to the managed clusters.

2.4.6.3. Role policy sample

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

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

2.4.7. Role binding policy

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.

2.4.7.1. Role binding policy YAML structure

```
apiVersion: policy.open-cluster-management.io/v1
kind: Policy
metadata:
  name:
  namespace:
spec:
  complianceType:
  remediationAction:
```

```
namespaces:
  exclude:
  include:
object-templates:
  - complianceType:
    objectDefinition:
      kind: RoleBinding # role binding must exist
      apiVersion: rbac.authorization.k8s.io/v1
      metadata:
        name: operate-pods-rolebinding
      subjects:
      - kind: User
        name: admin # Name is case sensitive
        apiGroup:
      roleRef:
        kind: Role #this must be Role or ClusterRole
        name: operator # this must match the name of the Role or ClusterRole you wish to bind to
        apiGroup: rbac.authorization.k8s.io
  ...
```

2.4.7.2. Role binding policy table

Field	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 to identify the policy resource.
metadata.namespaces	Required. The namespace within the managed cluster where the policy is created.
spec	Required. Specifications of how compliance violations are identified and fixed.
metadata.name	Required. The name for identifying the policy resource.
metadata.namespaces	Optional.
spec.complianceType	Required. Set the value to **"musthave"**

Field	Description
spec.namespace	Required. Managed cluster namespace to which you want to apply the policy. Enter parameter values for **include**, which are the namespaces you want to apply to the policy to. The **exclude** parameter specifies the namespaces you explicitly do not want to apply the policy to. **Note**: A namespace that is specified in the object template of a policy controller overrides the namespace in the corresponding parent policy.
spec.remediationAction	Required. Specifies the remediation of your policy. The parameter values are **enforce** and **inform**. **Important**: Some policies might not support the enforce feature.
spec.object-template	Required. Used to list any other Kubernetes object that must be evaluated or applied to the managed clusters.

2.4.7.3. Role binding policy sample

See **policy-rolebinding.yaml** to view the policy sample. Learn how to manage a role binding policy, see Managing role binding policies for more details. Refer to Kubernetes configuration policy controller to learn about other configuration policies. See Manage security policies to manage other policies.

2.4.8. Security Context Constraints policy

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.

2.4.8.1. SCC policy YAML structure

```
apiVersion: policy.open-cluster-management.io/v1
kind: Policy
metadata:
  name: policy-scc
  namespace: open-cluster-management-policies
spec:
  complianceType:
  remediationAction:
  namespaces:
    exclude:
    include:
  object-templates:
    - complianceType:
      objectDefinition:
        apiVersion:
        kind: SecurityContextConstraints # restricted scc
      metadata:
        annotations:
          kubernetes.io/description:
        name: sample-restricted-scc
      allowHostDirVolumePlugin:
      allowHostIPC:
      allowHostNetwork:
      allowHostPID:
      allowHostPorts:
      allowPrivilegeEscalation:
      allowPrivilegedContainer:
      allowedCapabilities:
      defaultAddCapabilities:
      fsGroup:
       type:
      groups:
      - system:
      priority:
      readOnlyRootFilesystem:
      requiredDropCapabilities:
      runAsUser:
        type:
      seLinuxContext:
        type:
      supplementalGroups:
        type:
      users:
      volumes:
```

2.4.8.2. SCC policy table

Field	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 to identify the policy resource.
metadata.namespace	Required. The namespace within the managed cluster where the policy is created.
spec.complianceType	Required. Set the value to **"musthave"**
spec.remediationAction	Required. Specifies the remediation of your policy. The parameter values are **enforce** and **inform**. **Important**: Some policies might not support the enforce feature.

Field	Description
spec.namespace	Required. Managed cluster namespace to which you want to apply the policy. Enter parameter values for **include**, which are the namespaces you want to apply to the policy to. The **exclude** parameter specifies the namespaces you explicitly do not want to apply the policy to. **Note**: A namespace that is specified in the object template of a policy controller overrides the namespace in the corresponding parent policy.
spec.object-template	Required. Used to list any other Kubernetes object 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.

2.4.8.3. SCC policy sample

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

See **policy-scc.yaml** to view the policy sample. To learn how to manage an SCC policy, see Managing Security Context Constraints policies for more details.

Refer to Kubernetes configuration policy controller to learn about other configuration policies. See Manage security policies to manage other policies.

2.4.9. ETCD encryption policy

Apply the **etcd-encryption** policy to detect, or enable encryption of sensitive data in the ETCD data-store. 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:

2.4.9.1. ETCD encryption policy YAML structure

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

```
apiVersion: policy.open-cluster-management.io/v1
kind: ConfigurationPolicy
metadata:
  name: policy-etcdencryption
  namespace:
spec:
  complianceType:
```

```
remediationAction:
namespaces:
  exclude:
  include:
object-templates:
  - complianceType:
    objectDefinition:
      apiVersion: config.openshift.io/v1
      kind: APIServer
      metadata:
        name: cluster
      spec:
        encryption:
          type:
      ...
```

2.4.9.2. ETCD encryption policy table

Table 2.4. Parameter table

Field	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, for example, **ConfigurationPolicy**.
metadata.name	Required. The name for identifying the policy resource.
metadata.namespaces	Optional.
spec.namespace	Required. The namespaces within the hub cluster that the policy is applied to. Enter parameter values for **include**, which are the namespaces you want to apply to the policy to. The **exclude** parameter specifies the namespaces you explicitly do not want to apply the policy to. **Note**: A namespace that is specified in the object template of a policy controller overrides the namespace in the corresponding parent policy.
remediationAction	Optional. Specifies the remediation of your policy. The parameter values are **enforce** and **inform**. **Important**: Some policies might not support the enforce feature.
disabled	Required. Set the value to **true** or **false**. The **disabled** parameter provides the ability to enable and disable your policies.

Field	Description
spec.complianceType	Required. Set the value to **"musthave"**
spec.object-template	Optional. Used to list any other Kubernetes object that must be evaluated or applied to the managed clusters. See Encrypting etcd data in the OpenShift Container Platform documentation.

2.4.9.3. Etcd encryption policy sample

See **policy-etcdencryption.yaml** for the policy sample. View Managing ETCD encryption policies for more information. Refer to Kubernetes configuration policy controller to view other configuration policies that are monitored by the controller.

2.4.10. Integrating gatekeeper constraints and constraint templates

Gatekeeper is a validating webhook that enforces custom resource definition (CRD) 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 policy can be used to evaluate Kubernetes resource compliance. You can leverage a OPA as the policy engine, and use Rego as the policy language.

The gatekeeper policy is created as a Kubernetes configuration policy in Red Hat Advanced Cluster Management. Gatekeeper policies include constraint templates (**ConstraintTemplates**) and **Constraints**, audit templates, and admission templates. For more information, see the Gatekeeper upstream repository.

Red Hat Advanced Cluster Management applies the following constraint templates in your Red Hat Advanced Cluster Management gatekeeper policy:

- **ConstraintTemplates** and constraints: Use the **policy-gatekeeper-k8srequiredlabels** policy to create a gatekeeper constraint template on the managed cluster.

```
apiVersion: policy.open-cluster-management.io/v1
kind: ConfigurationPolicy
metadata:
  name: policy-gatekeeper-k8srequiredlabels
spec:
  remediationAction: enforce # will be overridden by remediationAction in parent policy
  severity: low
  object-templates:
    - complianceType: musthave
      objectDefinition:
        apiVersion: templates.gatekeeper.sh/v1beta1
        kind: ConstraintTemplate
        metadata:
          name: k8srequiredlabels
        spec:
          crd:
            spec:
              names:
                kind: K8sRequiredLabels
              validation:
                # Schema for the `parameters` field
                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])
                }
    - complianceType: musthave
      objectDefinition:
        apiVersion: constraints.gatekeeper.sh/v1beta1
        kind: K8sRequiredLabels
        metadata:
          name: ns-must-have-gk
        spec:
          match:
            kinds:
              - apiGroups: [""]
                kinds: ["Namespace"]
            namespaces:
              - e2etestsuccess
              - e2etestfail
          parameters:
            labels: ["gatekeeper"]
```

- audit template: Use the **policy-gatekeeper-audit** to periodically check and evaluate existing resources against the gatekeeper policies that are enforced to detect existing miscongfigurations.

```
apiVersion: policy.open-cluster-management.io/v1
kind: ConfigurationPolicy
metadata:
  name: policy-gatekeeper-audit
spec:
  remediationAction: inform # will be overridden by remediationAction in parent policy
  severity: low
  object-templates:
    - complianceType: musthave
      objectDefinition:
        apiVersion: constraints.gatekeeper.sh/v1beta1
        kind: K8sRequiredLabels
        metadata:
          name: ns-must-have-gk
        status:
          totalViolations: 0
```

- admission template: Use the **policy-gatekeeper-admission** to check for misconfigurations that are created 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
```

See **policy-gatekeeper-sample.yaml** for more details.

Learn how to use Red Hat Advanced Cluster Management gatekeeper operator policy to install gatekeeper and create a Red Hat Advanced Cluster Management gatekeeper operator policy, see Creating a gatekeeper policy from the console for more details. Refer to Governance and risk for more topics on the security framework.

2.4.11. Compliance operator policy

Compliance operator is an operator that runs OpenSCAP and allows you to keep your Red Hat OpenShift Container Platform cluster compliant with the security benchmark that you need. 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 4.7 and 4.6, support the compliance operator policy. For more information, see *Understanding the Compliance Operator* in the OpenShift Container Platform documentation for more details.

2.4.11.1. 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:

```
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: "4.7"
          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**.

You can also create and apply the E8 scan policy after you have installed the compliance operator. For more information, see E8 scan policy.

To learn about managing compliance operator policies, see Managing compliance operator policies for more details. Refer to Kubernetes configuration policy controller for more topics about configuration policies.

2.4.12. 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 4.7 and 4.6, support the E8 scan policy. For more information, see Understanding the Compliance Operator in the OpenShift Container Platform documentation for more details.

2.4.12.1. 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
```

See a sample of the **policy-compliance-operator-e8-scan.yaml** file. For more information on creating an E8 scan policy, see Managing E8 scan policies.

2.5. MANAGE SECURITY POLICIES

Use the Governance and risk dashboard to create, view, and manage your security policies and policy violations. You can create YAML files for your policies from the CLI and console.

From the *Governance and risk* page, you can customize your Summary view by filtering the violations by categories or standards, collapse the summary to see less information, and you can search for policies. You can also filter the violation table view by policies or cluster violations.

The table of policies list the following details of a policy: *Policy name*, *Namespace*, *Remediation*, *Cluster violation*, *Standards*, *Categories*, and *Controls*. You can edit, disable, inform or remove a policy by selecting the **Actions** icon.

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, Placement details, and a table list of *Policy templates*.
- *Status*: Select the *Status* tab to view a table list of violations. You can filter your view by *Clusters* or *Templates*. To view the compliance status of your policy, select the *Status* tab. Click the *View history* link to view a list of violation messages.
- *YAML*: Select the *YAML* tab to view, and or edit your policy with the editor. Select the YAML toggle to view or hide the editor.

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

- Managing security policies
- Managing configuration policies
- Managing image vulnerability policies
- Managing memory usage policies
- Managing namespace policies
- Managing pod policies
- Managing pod security policies
- Managing role policies
- Managing role binding policies
- Managing Security Context Constraints policies
- Managing certificate policies
- Managing IAM policies
- Managing ETCD encryption policies
- Managing gatekeeper policies
- Managing compliance operator policies
- Managing E8 scan policies

Refer to Governance and risk for more topics.

2.5.1. Deploy policies using GitOps

You can deploy a set of policies across a fleet of managed clusters with the governance framework. You can add to the open source community, **policy-collection** by contributing to and using the policies in the repository. For more information, see Contributing a custom policy. Policies in each of the **stable** and **community** folders from the open source community are further organized according to NIST Special Publication 800-53.

Continue reading to learn best practices to use GitOps to automate and track policy updates and creation through a Git repository.

Prerequsite: Before you begin, be sure to fork the **policy-collection** repository.

2.5.1.1. Customizing your local repository

Customize your local repository by consolidating the **stable** and **community** policies into a single folder. Remove the policies you do not want to use. Complete the following steps to customize your local repository:

1. Create a new directory in the repository to hold the policies that you want to deploy. Be sure that you are in your local **policy-collection** repository on your main default branch for GitOps. Run the following command:

   ```
   mkdir my-policies
   ```

2. Copy all of the **stable** and **community** policies into your **my-policies** directory. Start with the **community** policies first, in case the **stable** folder contains duplicates of what is available in the community. Run the following commands:

   ```
   cp -R community/* my-policies/
   cp -R stable/* my-policies/
   ```

   Now that you have all of the policies in a single parent directory structure, you can edit the policies in your fork.

   Tips:

   - It is best practice to remove the policies you are not planning to use.
   - Learn about policies and the definition of the policies from the following list:
     - Purpose: Understand what the policy does.
     - Remediation Action: Does the policy only inform you of compliance, or enforce the policy and make changes? See the **spec.remediationAction** parameter. If changes are enforced, make sure you understand the functional expectation. Remember to check which policies support enforcement. For more information, view the *Validate* section.
       Note: The **spec.remediationAction** set for the policy overrides any remediation action that is set in the individual **spec.policy-templates**.
     - Placement: What clusters is the policy deployed to? By default, most policies target the clusters with the **environment: dev** label. Some policies may target OpenShift Container Platform clusters or another label. You can update or add additional labels to include other clusters. When there is no specific value, the policy is applied to all of your clusters. You can also create multiple copies of a policy and customize each one if you want to use a policy that is configured one way for one set of clusters and configured another way for another set of clusters.

2.5.1.2. Committing to your local repository

After you are satisfied with the changes you have made to your directory, commit and push your changes to Git so that they can be accessed by your cluster.

Note: This example is used to show the basics of how to use policies with GitOps, so you might have a different workflow to get changes to your branch.

Complete the following steps:

1. From your terminal, run **git status** to view your recent changes in your directory that you previously created. Add your new directory to the list of changes to be committed with the following command:

   ```
   git add my-policies/
   ```

2. Commit the changes and customize your message. Run the following command:

   ```
   git commit -m “Policies to deploy to the hub cluster”
   ```

3. Push the changes to the branch of your forked repository that is used for GitOps. Run the following command:

   ```
   git push origin <your_default_branch>master
   ```

Your changes are committed.

2.5.1.3. Deploying policies to your cluster

After you push your changes, you can deploy the policies to your Red Hat Advanced Cluster Management for Kubernetes installation. Post deployment, your hub cluster is conncted to your Git repository. Any further changes to your chosen branch of the Git repository is reflected in your cluster.

The **deploy.sh** script creates **Channel** and **Subscription** resources in your hub cluster. The channel connects to the Git repository, and the subscription specifies the data to bring to the cluster through the channel. As a result, all policies defined in the specified subdirectory are created on your hub.

After the policies are created by the subscription, Red Hat Advanced Cluster Management analyzes the policies and creates additional policy resources in the namespace associated with each managed cluster that the policy is applied to, based on the defined placement rule.

The policy is then copied to the managed cluster from its respective managed cluster namespace on the hub cluster. As a result, the policies in your Git repository are pushed to all managed clusters that have labels that match the **clusterSelector** that are defined in the placement rule of your policy.

Complete the following steps:

1. From the **policy-collection** folder, run the following command to change the directory:

   ```
   cd deploy
   ```

2. Make sure that your command line interface (CLI) is configured to create resources on the correct cluster with the following command:

   ```
   oc cluster-info
   ```

The output of the command displays the API server details for the cluster, where Red Hat Advanced Cluster Management is installed. If the correct URL is not displayed, configure your CLI to point to the correct cluster. See Using the OpenShift CLI for more information.

3. Create a namespace where your policies are created to control access and to organize the policies. Run the following command:

```
oc create namespace policy-namespace
```

4. Run the following command to deploy the policies to your cluster:

```
./deploy.sh -u https://github.com/<your-repository>/policy-collection -p my-policies -n policy-
namespace
```

Replace **your-repository** with your Git user name or repository name.

Note: For reference, the full list of arguments for the **deploy.sh** script uses the following syntax:

```
./deploy.sh [-u <url>] [-b <branch>] [-p <path/to/dir>] [-n <namespace>] [-a|--name
<resource-name>]
```

View the following explanations for each argument:

- URL: The URL to the repository that you forked from the main **policy-collection** repository. The default URL is **https://github.com/stolostron/policy-collection.git**.
- Branch: Branch of the Git repository to point to. The default branch is **main**.
- Subdirectory Path: The subdirectory path you created to contain the policies you want to use. In the previous sample, we used the **my-policies** subdirectory, but you can also specify which folder you want start with. For example, you can use **my-policies/AC-Access-Control**. The default folder is **stable**.
- Namespace: The namespace where the resources and policies are created on the hub cluster. These instructions use the **policy-namespace** namespace. The default namespace is **policies**.
- Name Prefix: Prefix for the **Channel** and **Subscription** resources. The default is **demo-stable-policies**. After you run the **deploy.sh** script, any user with access to the repository can commit changes to the branch, which pushes changes to exisiting policies on your clusters.

2.5.1.4. Verifying GitOps policy deployments from the console

Verify that your changes were applied to your policies from the console. You can also make more changes to your policy from the console. Complete the following steps:

1. Log in to your Red Hat Advanced Cluster Management cluster.
2. From the navigation menu, select **Govern risk**.
3. Check for the following policy details:
   - Why is a specific policy compliant or non-compliant on the clusters that it was distributed to?

- Are the policies applied to the correct clusters?
- If this policy is not distributed to any clusters, why?

4. Identify the GitOps deployed policies that you created or modified. The GitOps deployed policies can be identified by the annotation that is applied automatically. Annotations for the GitOps deployed policies resemble the following paths:

```
apps.open-cluster-management.io/hosting-deployable: policies/deploy-stable-policies-Policy-policy-role9

apps.open-cluster-management.io/hosting-subscription: policies/demo-policies

apps.open-cluster-management.io/sync-source: subgbk8s-policies/demo-policies
```

GitOps annotations are valuable to see which subscription created the policy. You can also add your own labels to your policies so that you can write runtime queries that select policies based on labels.

For example, you can add a label to a policy with the following command:

```
oc label policy <policy-name> -n <policy-namespace> <key>=<value>
```

Then, you can query policies that have labels with the following command:

```
oc get policies -n <policy-namespace> -l <key>=<value>
```

Your policies are deployed using GitOps.

2.5.2. Managing security policies

Create a security policy to report and validate your cluster compliance based on your specified security standards, categories, and controls. To create a policy for Red Hat Advanced Cluster Management for Kubernetes, you must create a YAML file on your managed clusters.

Note: You can copy and paste an existing policy in to the *Policy YAML*. The values for the parameter fields are automatically entered when you paste your existing policy. You can also search the contents in your policy YAML file with the search feature.

2.5.2.1. Creating a security policy

You can create a security policy from the command line interface (CLI) or from the console. Cluster administrator access is required.

Important: You must define a PlacementPolicy and PlacementBinding to apply your policy to a specific cluster. Enter a value for the *Cluster selector* field to define a PlacementPolicy and PlacementBinding. View the definitions of the objects that are required for your Red Hat Advanced Cluster Management policy:

- *PlacementRule*: Defines a *cluster selector* where the policy must be deployed.
- *PlacementBinding*: Binds the placement to a PlacementPolicy.

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

2.5.2.1.1. 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:

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

2. Define the template that the policy uses. Edit your **.yaml** file by adding a **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
     namespaces:
       include: ["default"]
       exclude: ["kube*"]
     policy-templates:
       - objectDefinition:
           apiVersion: policy.open-cluster-management.io/v1
           kind: ConfigurationPolicy
           metadata:
             namespace: kube-system # will be inferred
             name: operator
           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 **PlacementRule**. Be sure to change the **PlacementRule** to specify the clusters where the policies need to be applied, either by **clusterNames**, or **clusterLabels**. View Creating and managing placement rules. Your **PlacementRule** might resemble the following content:

   ```
   apiVersion: apps.open-cluster-management.io/v1
   kind: PlacementRule
   metadata:
     name: placement1
   spec:
     clusterConditions:
   ```

```
    - type: ManagedClusterConditionAvailable
      status: "True"
  clusterNames:
  - "cluster1"
  - "cluster2"
  clusterLabels:
    matchLabels:
      cloud: IBM
```

4. Define a **PlacementBinding** to bind your policy and your **PlacementRule**. 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.mcm.ibm.com
  kind: Policy
```

2.5.2.1.1.1. 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:

```
kubectl get securityepolicy <policy-name> -n <namespace> -o yaml
```

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

```
kubectl describe securitypolicy <name> -n <namespace>
```

2.5.2.1.2. Creating a cluster security policy from the console

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

1. From the navigation menu, click **Govern risk**.
2. To create a policy, click **Create policy**.
3. Enter or select values for the following parameters:
   - Name
   - Specifications
   - Cluster selector
   - Remediation action

- Standards
- Categories
- Controls

4. View the following example Red Hat Advanced Cluster Management for Kubernetes security policy definition. Copy and paste the YAML file for your policy.
   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'
spec:
  complianceType: musthave
  namespaces:
    exclude: ["kube*"]
    include: ["default"]
  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

---
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.mcm.ibm.com

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

```
kind: PlacementRule
metadata:
  name: placement-pod
spec:
  clusterConditions:
    - type: ManagedClusterConditionAvailable
      status: "True"
  clusterLabels:
    matchLabels:
      cloud: "IBM"
```

5. Click **Create Policy**.

A security policy is created from the console.

2.5.2.1.2.1. Viewing your security policy from the console

You can view any security policy and its status from the console.

1. Log in to your cluster from the console.
2. From the navigation menu, click **Governance and risk** 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.
3. Select one of your policies to view more details. The *Overview* tab, *Status* tab, and *YAML* tab are displayed.
   When the cluster or policy status cannot be determined, the following message is displayed: **No status**.

2.5.2.2. Updating security policies

Learn to update security policies by viewing the following section.

2.5.2.2.1. Disabling security policies

Your policy is enabled by default. You can disable your policy by completing the following steps:

1. Log in to your Red Hat Advanced Cluster Management for Kubernetes console.
2. From the navigation menu, click **Govern risk** to view a table list of your policies.
3. Disable your policy by clicking the **Actions** icon > **Disable policy**. The *Disable Policy* dialog box appears.
4. Click **Disable policy**.

Your policy is disabled.

2.5.2.2.2. 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:

```
kubectl delete policy <securitypolicy-name> -n <open-cluster-management-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: **kubectl get policy <securitypolicy-name> -n <open-cluster-management-namespace>**

- Delete a security policy from the console:
  a. From the navigation menu, click **Govern risk** to view a table list of your policies.
  b. Click the **Actions** icon for the policy you want to delete in the policy violation table.
  c. Click **Remove**.
  d. From the *Remove policy* dialog box, click **Remove policy**

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

2.5.3. Managing configuration policies

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

2.5.3.1. Creating a configuration policy

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

2.5.3.1.1. 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:

```
kubectl 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: kube-system
spec:
  namespaces:
    include: ["default", "kube-*"]
    exclude: ["kube-system"]
  remediationAction: inform
  disabled: false
  complianceType: musthave
  object-templates:
  ...
```

2. Apply the policy by running the following command:

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

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

```
kubectl get policy --namespace=<namespace>
```

Your configuration policy is created.

2.5.3.1.1.1. Viewing your configuration policy from the CLI

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

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

```
kubectl get policy <policy-name> -n <namespace> -o yaml
```

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

```
kubectl describe policy <name> -n <namespace>
```

2.5.3.1.2. 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. Complete the following steps to create a configuration policy from the console:

1. Log in to your cluster from the console.
2. From the navigation menu, click **Governance and risk**.
3. Click **Create policy**.
4. Specify the policy you want to create by selecting one of the configuration policies for the specification parameter. Continue to enter or select the appropriate values for the following fields:
   - Name
   - Specifications
   - Cluster selector
   - Remediation action
   - Standards
   - Categories
   - Controls
5. Click **Create**.

2.5.3.1.2.1. Viewing your configuration policy from the console

You can view any configuration policy and its status from the console.

1. Log in to your cluster from the console.
2. From the navigation menu, click **Govern risk** 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.
3. Select one of your policies to view more details. The *Overview* tab, *Status* tab, and *YAML* tab are displayed.

2.5.3.2. Updating configuration policies

Learn to update configuration policies by viewing the following section.

2.5.3.2.1. Disabling configuration policies

Complete the following steps to disable your configuration policy:

1. Log in to your Red Hat Advanced Cluster Management for Kubernetes console.
2. From the navigation menu, click **Govern risk** to view a table list of your policies.
3. Disable your policy by clicking the **Actions** icon > **Disable**. The *Disable Policy* dialog box appears.
4. Click **Disable policy**.

Your policy is disabled.

2.5.3.3. Deleting a configuration policy

Delete a configuration policy from the CLI or the console.

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

     ```
     kubectl delete policy <policy-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:

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

- Delete a configuration policy from the console:
  a. From the navigation menu, click **Govern risk** to view a table list of your policies.
  b. Click the **Actions** icon for the policy you want to delete in the policy violation table.
  c. Click **Remove**.
  d. From the *Remove policy* dialog box, click **Remove policy**.

Your policy is deleted.

See configuration policy samples that are supported by Red Hat Advanced Cluster Management from the CM-Configuration-Management folder.

Alternatively, you can refer to Kubernetes configuration policy controller to view other configuration policies that are monitored by the controller. For details to manage other policies, refer to Managing security policies.

2.5.4. Managing image vulnerability policies

Configuration policy controller monitors the status of image vulnerability policies. Image vulnerability policies are applied to check if your containers have vulnerabilities. Learn to create, apply, view, and update your image vulnerability policy.

2.5.4.1. Creating an image vulnerability policy

You can create a YAML for your image vulnerability policy from the command line interface (CLI) or from the console. View the following sections to create an image vulnerability policy:

2.5.4.1.1. Creating an image vulnerability policy from the CLI

Complete the following steps to create an image vulnerability policy from the CLI:

1. Create a YAML file for your image vulnerability policy by running the following command:

```
kubectl create -f imagevulnpolicy-1.yaml
```

2. Apply the policy by running the following command:

```
kubectl apply -f <imagevuln-policy-file-name>  --namespace=<namespace>
```

3. List and verify the policies by running the following command:

```
kubectl get imagevulnpolicy --namespace=<namespace>
```

Your image vulnerability policy is created.

2.5.4.1.1.1. Viewing your image vulnerability policy from the CLI

Complete the following steps to view your image vulnerability policy from the CLI:

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

```
kubectl get imagevulnpolicy <policy-name> -n <namespace> -o yaml
```

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

```
kubectl describe imagevulnpolicy <name> -n <namespace>
```

2.5.4.2. Creating an image vulnerability policy from the console

As you create an image vulnerability policy from the console, a YAML file is also created in the YAML editor. Complete the following steps to create the image vulnerability policy from the console:

1. Log in to your cluster from the console.
2. From the navigation menu, click **Governance and risk**.
3. Click **Create policy**.
4. Select **ImageManifestVulnPolicy** from the *Specifications* field. Parameter values are automatically set. You can edit your values.
5. Click **Create**.

An image vulnerability policy is created.

2.5.4.3. Viewing image vulnerability violations from the console

1. From the navigation menu, click **Govern risk** to view a table list of your policies.
2. Select **policy-imagemanifestvulnpolicy** > ***Status*** to view the cluster location of the violation. Your image vulnerability violation might resemble the following:

```
imagemanifestvulns exist and should be deleted:
[sha256.7ac7819e1523911399b798309025935a9968b277d86d50e5255465d6592c0266] in
namespace default;
[sha256.4109631e69d1d562f014dd49d5166f1c18b4093f4f311275236b94b21c0041c0] in
namespace calamari;
[sha256.573e9e0a1198da4e29eb9a8d7757f7afb7ad085b0771bc6aa03ef96dedc5b743,
sha256.a56d40244a544693ae18178a0be8af76602b89abe146a43613eaeac84a27494e,
sha256.b25126b194016e84c04a64a0ad5094a90555d70b4761d38525e4aed21d372820] in
namespace open-cluster-management-agent-addon;
[sha256.64320fbf95d968fc6b9863581a92d373bc75f563a13ae1c727af37450579f61a] in
namespace openshift-cluster-version
```

3. Navigate to your OpenShift Container Platform console by selecting the *Cluster* link.
4. From the navigation menu on the OpenShift Container Platform console, click **Administration** > **Custom Resource Definitions**.
5. Select **imagemanifestvulns** > *Instances* **tab** to view all of the **imagemanifestvulns** instances.
6. Select an entry to view more details.

2.5.4.4. Updating image vulnerability policies

Learn to update image vulnerability policies by viewing the following section.

2.5.4.4.1. Disabling image vulnerability policies

Complete the following steps to disable your image vulnerability policy:

1. Log in to your Red Hat Advanced Cluster Management for Kubernetes console.
2. From the navigation menu, click **Govern risk** to view a table list of your policies.
3. Disable your policy by clicking the **Actions** icon > **Disable**. The *Disable Policy* dialog box appears.
4. Click **Disable policy**.

Your policy is disabled.

2.5.4.4.2. Deleting an image vulnerability policy

Delete the image vulnerability policy from the CLI or the console.

- Delete an image vulnerability policy from the CLI:
    a. Delete a certificate policy by running the following command:

    ```
    kubectl delete policy <imagevulnpolicy-name> -n <namespace>
    ```

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

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

    ```
    kubectl get policy <imagevulnpolicy-name> -n <namespace>
    ```

- Delete an image vulnerability policy from the console:
    a. From the navigation menu, click **Govern risk** to view a table list of your policies.
    b. Click the **Actions** icon for the policy you want to delete in the policy violation table.
    c. Click **Remove**.
    d. From the *Remove policy* dialog box, click **Remove policy**.

Your image vulnerability policy is deleted.

View a sample of an image vulnerability policy, see *Image vulnerability policy sample* from the Image vulnerability policy page. See Kubernetes configuration policy controller to learn about other policies that are monitored by the Kubernetes configuration policy controller. See Managing security policies to manage other policies.

2.5.5. Managing memory usage policies

Apply a memory usage policy to limit or restrict your memory and compute usage. Learn to create, apply, view, and update your memory usage policy in the following sections.

2.5.5.1. Creating a memory usage policy

You can create a YAML file for your memory usage policy from the command line interface (CLI) or from the console. View the following sections to create a memory usage policy:

2.5.5.1.1. Creating a memory usage policy from the CLI

Complete the following steps to create a memory usage policy from the CLI:

1. Create a YAML file for your memory usage policy by running the following command:

    ```
    kubectl create -f memorypolicy-1.yaml
    ```

2. Apply the policy by running the following command:

```
kubectl apply -f <memory-policy-file-name>  --namespace=<namespace>
```

3. List and verify the policies by running the following command:

```
kubectl get memorypolicy --namespace=<namespace>
```

Your memory usage policy is created from the CLI.

2.5.5.1.1.1. Viewing your policy from the CLI

Complete the following steps to view your memory usage policy from the CLI:

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

```
kubectl get memorypolicy <policy-name> -n <namespace> -o yaml
```

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

```
kubectl describe memorypolicy <name> -n <namespace>
```

2.5.5.1.2. Creating an memory usage policy from the console

As you create a memory usage policy from the console, a YAML file is also created in the YAML editor. Complete the following steps to create the memory usage policy from the console:

1. Log in to your Red Hat Advanced Cluster Management for Kubernetes console.
2. From the navigation menu, click **Governance and risk**.
3. Click **Create policy**.
4. Select **Limitrange** from the *Specifications* field. Parameter values are automatically set. You can edit your values.
5. Click **Create**.

2.5.5.1.2.1. Viewing your memory usage policy from the console

You can view any memory usage policy and its status from the console.

1. Log in to your cluster from the console.
2. From the navigation menu, click **Govern risk** 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.
3. Select one of your policies to view more details.
4. View the policy violations by selecting the *Status* tab.

2.5.5.2. Updating memory usage policies

Learn to update memory usage policies by viewing the following section.

2.5.5.2.1. Disabling memory usage policies

Complete the following steps to disable your memory usage policy:

1. Log in to your Red Hat Advanced Cluster Management for Kubernetes console.
2. From the navigation menu, click **Govern risk** to view a table list of your policies.
3. Disable your policy by clicking the **Actions** icon > **Disable**. The *Disable Policy* dialog box appears.
4. Click **Disable policy**.

Your policy is disabled.

2.5.5.2.2. Deleting a memory usage policy

Delete the memory usage policy from the CLI or the console.

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

     ```
     kubectl delete policy <memorypolicy-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:

     ```
     kubectl get policy <memorypolicy-name> -n <namespace>
     ```

- Delete a memory usage policy from the console:
  a. From the navigation menu, click **Govern risk** to view a table list of your policies.
  b. Click the **Actions** icon for the policy you want to delete in the policy violation table.
  c. Click **Remove**.
  d. From the *Remove policy* dialog box, click **Remove policy**.

Your memory usage policy is deleted.

View a sample of a memory usage policy, see *Memory usage policy sample* from the Memory usage policy page. See Kubernetes configuration policy controller to learn about other configuration policies. See Managing security policies to manage other policies.

2.5.6. Managing namespace policies

Namespace policies are applied to define specific rules for your namespace. Learn to create, apply, view, and update your namespace policy in the following sections.

2.5.6.1. Creating a namespace policy

You can create a YAML file for your namespace policy from the command line interface (CLI) or from the console. View the following sections to create a namespace policy:

2.5.6.1.1. Creating a namespace policy from the CLI

Complete the following steps to create a namespace policy from the CLI:

1. Create a YAML file for your namespace policy by running the following command:

   ```
   kubectl create -f namespacepolicy-1.yaml
   ```

2. Apply the policy by running the following command:

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

3. List and verify the policies by running the following command:

   ```
   kubectl get namespacepolicy --namespace=<namespace>
   ```

Your namespace policy is created from the CLI.

2.5.6.1.1.1. Viewing your namespace policy from the CLI

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

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

   ```
   kubectl get namespacepolicy <policy-name> -n <namespace> -o yaml
   ```

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

   ```
   kubectl describe namespacepolicy <name> -n <namespace>
   ```

2.5.6.1.2. Creating a namespace policy from the console

As you create a namespace policy from the console, a YAML file is also created in the YAML editor. Complete the following steps to create a namespace policy from the console:

1. Log in to your Red Hat Advanced Cluster Management for Kubernetes console.
2. From the navigation menu, click **Governance and risk**.
3. Click **Create policy**.
4. Select **Namespace** from the *Specifications* field. Parameter values are automatically set. You can edit your values.
5. Click **Create**.

2.5.6.1.2.1. Viewing your namespace policy from the console

You can view any namespace policy and its status from the console.

1. Log in to your cluster from the console.
2. From the navigation menu, click **Governance and risk** 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.

3. Select one of your policies to view more details.

4. View the policy violations by selecting the *Status* tab.

2.5.6.2. Updating namespace policies

Learn to update namespace policies by viewing the following section.

2.5.6.2.1. Disabling namespace policies

Complete the following steps to disable your namespace policy:

1. Log in to your Red Hat Advanced Cluster Management for Kubernetes console.

2. From the navigation menu, click **Govern risk** to view a table list of your policies.

3. Disable your policy by clicking the **Actions** icon > **Disable**. The *Disable Policy* dialog box appears.

4. Click **Disable policy**.

Your policy is disabled.

2.5.6.2.2. Deleting a namespace policy

Delete a namespace policy from the CLI or the console.

- Delete a namespace policy from the CLI:

  a. Delete a namespace policy by running the following command:

  ```
  kubectl delete policy <namespacepolicy-name> -n <namespace>
  ```

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

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

  ```
  kubectl get policy <namespacepolicy-name> -n <namespace>
  ```

- Delete a namespace policy from the console:

  a. From the navigation menu, click **Govern risk** to view a table list of your policies.

  b. Click the **Actions** icon for the policy you want to delete in the policy violation table.

  c. Click **Remove**.

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

Your namespace policy is deleted.

View a sample of a namespace policy, see *Namespace policy sample* on the Namespace policy page. See Kubernetes configuration policy controller to learn about other configuration policies. See Managing security policies to manage other policies.

2.5.7. Managing pod policies

Kubernetes configuration policy controller monitors the status of you pod policies. Pod policies are applied to define the container rules for your pods. Learn to create, apply, view, and update your pod policy.

2.5.7.1. Creating a pod policy

You can create a YAML for your pod policy from the command line interface (CLI) or from the console. View the following sections to create a pod policy:

2.5.7.1.1. Creating a pod policy from the CLI

Complete the following steps to create a pod policy from the CLI:

1. Create a YAML file for your pod policy by running the following command:

   ```
   kubectl create -f podpolicy-1.yaml
   ```

2. Apply the policy by running the following command:

   ```
   kubectl apply -f <pod-policy-file-name>  --namespace=<namespace>
   ```

3. List and verify the policies by running the following command:

   ```
   kubectl get podpolicy --namespace=<namespace>
   ```

Your image pod policy is created from the CLI.

2.5.7.1.1.1. Viewing your policy from the CLI

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

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

   ```
   kubectl get podpolicy <policy-name> -n <namespace> -o yaml
   ```

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

   ```
   kubectl describe podpolicy <name> -n <namespace>
   ```

2.5.7.2. Creating an pod policy from the console

As you create a pod policy from the console, a YAML file is also created in the YAML editor. Complete the following steps to create the pod policy from the console:

1. Log in to your Red Hat Advanced Cluster Management for Kubernetes console.
2. From the navigation menu, click **Govern risk**.
3. Click **Create policy**.

4. Select **Pod** from the *Specifications* field. Parameter values are automatically set. You can edit your values.
5. Click **Create**.

Viewing your pod policy from the console
You can view any pod policy and its status from the console.

1. Log in to your cluster from the console.
2. From the navigation menu, click **Govern risk** 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.
3. Select one of your policies to view more details.
4. View the policy violations by selecting the *Status* tab.

2.5.7.3. Updating pod policies

Learn to update pod policies by viewing the following section.

2.5.7.3.1. Disabling pod policies

Complete the following steps to disable your pod policy:

1. Log in to your Red Hat Advanced Cluster Management for Kubernetes console.
2. From the navigation menu, click **Govern risk** to view a table list of your policies.
3. Disable your policy by clicking the **Actions** icon > **Disable**. The *Disable Policy* dialog box appears.
4. Click **Disable policy**.

Your policy is disabled.

2.5.7.3.2. Deleting a pod policy

Delete the pod policy from the CLI or the console.

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

   ```
   kubectl delete policy <podpolicy-name> -n <namespace>
   ```

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

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

   ```
   kubectl get policy <podpolicy-name> -n <namespace>
   ```

- Delete a pod policy from the console:
   a. From the navigation menu, click **Govern risk** to view a table list of your policies.

b. Click the **Actions** icon for the policy you want to delete in the policy violation table.

c. Click **Remove**.

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

Your pod policy is deleted.

View a sample of a pod policy, see *Pod policy sample* from the Pod policy page. See Kubernetes configuration policy controller to learn about other configuration policies. See Managing security policies to manage other policies.

2.5.8. Managing pod security policies

Apply a pod security policy to secure pods and containers. Learn to create, apply, view, and update your pod security policy in the following sections.

2.5.8.1. Creating a pod security policy

You can create a YAML file for your pod security policy from the command line interface (CLI) or from the console. View the following sections to create a pod security policy:

2.5.8.1.1. Creating a pod security policy from the CLI

Complete the following steps to create a pod security from the CLI:

1. Create a YAML file for your pod security policy by running the following command:

```
kubectl create -f podsecuritypolicy-1.yaml
```

2. Apply the policy by running the following command:

```
kubectl apply -f <podsecurity-policy-file-name>  --namespace=<namespace>
```

3. List and verify the policies by running the following command:

```
kubectl get podsecuritypolicy --namespace=<namespace>
```

Your pod security policy is created from the CLI.

2.5.8.1.1.1. Viewing your pod security policy from the CLI

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

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

```
kubectl get podsecuritypolicy <policy-name> -n <namespace> -o yaml
```

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

```
kubectl describe podsecuritypolicy <name> -n <namespace>
```

2.5.8.1.2. Creating a pod security policy from the console

As you create a pod security policy from the console, a YAML file is also created in the YAML editor. Complete the following steps to create the pod security policy from the console:

1. Log in to your Red Hat Advanced Cluster Management for Kubernetes console.
2. From the navigation menu, click **Govern risk**.
3. Click **Create policy**.
4. Select **Podsecuritypolicy** from the *Specifications* field. Parameter values are automatically set. You can edit your values.
5. Click **Create**.

2.5.8.1.2.1. Viewing your pod security policy from the console

You can view any pod security policy and its status from the console.

1. Log in to your cluster from the console.
2. From the navigation menu, click **Govern risk** 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.
3. Select one of your policies to view more details.
4. View the policy violations by selecting the *Status* tab.

2.5.8.2. Updating pod security policies

Learn to update pod security policies by viewing the following section.

2.5.8.2.1. Disabling pod security policies

Complete the following steps to disable your pod security policy:

1. Log in to your Red Hat Advanced Cluster Management for Kubernetes console.
2. From the navigation menu, click **Govern risk** to view a table list of your policies.
3. Disable your policy by clicking the **Actions** icon > **Disable**. The *Disable Policy* dialog box appears.
4. Click **Disable policy**.

Your policy is disabled.

2.5.8.2.2. Deleting a pod security policy

Delete the pod security policy from the CLI or the console.

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

```
kubectl delete policy <podsecurity-policy-name> -n <namespace>
```

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

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

```
kubectl get policy <podsecurity-policy-name> -n <namespace>
```

- Delete a pod security policy from the console:
  a. From the navigation menu, click **Govern risk** to view a table list of your policies.
  b. Click the **Actions** icon for the policy you want to delete in the policy violation table.
  c. Click **Remove**.
  d. From the *Remove policy* dialog box, click **Remove policy**.

Your pod security policy is deleted.

View a sample of a pod security policy, see *Pod security policy sample* on the Pod security policy page. See Kubernetes configuration policy controller to learn about other configuration policies. See Managing security policies to manage other policies.

2.5.9. Managing role policies

Kubernetes configuration policy controller monitors the status of role policies. Apply a role policy to set rules and permissions for specific roles in your cluster. Learn to create, apply, view, and update your role policy in the following sections.

2.5.9.1. Creating a role policy

You can create a YAML file for your role policy from the command line interface (CLI) or from the console. View the following sections to create a role policy:

2.5.9.1.1. Creating a role policy from the CLI

Complete the following steps to create a role from the CLI:

1. Create a YAML file for your role policy by running the following command:

```
kubectl create -f rolepolicy-1.yaml
```

2. Apply the policy by running the following command:

```
kubectl apply -f <role-policy-file-name>  --namespace=<namespace>
```

3. List and verify the policies by running the following command:

```
kubectl get rolepolicy --namespace=<namespace>
```

Your role policy is created from the CLI.

2.5.9.1.1.1. Viewing your role policy from the CLI

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

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

   ```
   kubectl get rolepolicy <policy-name> -n <namespace> -o yaml
   ```

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

   ```
   kubectl describe rolepolicy <name> -n <namespace>
   ```

2.5.9.1.2. Creating a role policy from the console

As you create a role policy from the console, a YAML file is also created in the YAML editor. Complete the following steps to create the role policy from the console:

1. Log in to your Red Hat Advanced Cluster Management for Kubernetes console.
2. From the navigation menu, click **Govern risk**.
3. Click **Create policy**.
4. Select **Role** from the *Specifications* field. Parameter values are automatically set. You can edit your values.
5. Click **Create**.

2.5.9.1.2.1. Viewing your role policy from the console

You can view any role policy and its status from the console.

1. Log in to your cluster from the console.
2. From the navigation menu, click **Govern risk** 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.
3. Select one of your policies to view more details.
4. View the policy violations by selecting the *Status* tab.

2.5.9.2. Updating role policies

Learn to update role policies by viewing the following section.

2.5.9.2.1. Disabling role policies

Complete the following steps to disable your role policy:

1. Log in to your Red Hat Advanced Cluster Management for Kubernetes console.
2. From the navigation menu, click **Govern risk** to view a table list of your policies.
3. Disable your policy by clicking the **Actions** icon > **Disable**. The *Disable Policy* dialog box appears.
4. Click **Disable policy**.

Your policy is disabled.

2.5.9.2.2. Deleting a role policy

Delete the role policy from the CLI or the console.

- Delete a role policy from the CLI:

  a. Delete a role policy by running the following command:

  ```
  kubectl delete policy <podsecurity-policy-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:

  ```
  kubectl get policy <podsecurity-policy-name> -n <namespace>
  ```

- Delete a role policy from the console:

  a. From the navigation menu, click **Govern risk** to view a table list of your policies.

  b. Click the **Actions** icon for the policy you want to delete in the policy violation table.

  c. Click **Remove**.

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

Your role policy is deleted.

See the **policy-role.yaml** for the sample policy. Refer to Kubernetes configuration policy controller to view other configuration policies that are monitored by the controller.

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

2.5.10. Managing role binding policies

Learn to create, apply, view, and update your role binding policies.

2.5.10.1. Creating a role binding policy

You can create a YAML file for your role binding policy from the command line interface (CLI) or from the console. View the following sections to create a role binding policy:

2.5.10.1.1. Creating a role binding policy from the CLI

Complete the following steps to create a role binding policy from the CLI:

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

   ```
   kubectl create -f rolebindingpolicy.yaml
   ```

2. Apply the policy by running the following command:

   ```
   kubectl apply -f <rolebinding-policy-file-name>  --namespace=<namespace>
   ```

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

```
kubectl get rolebindingpolicy --namespace=<namespace>
```

Your role binding policy is created.

2.5.10.1.1.1. Viewing your role binding policy from the CLI

Complete the following steps to view your role binding policy from the CLI:

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

   ```
   kubectl get rolebindingpolicy <policy-name> -n <namespace> -o yaml
   ```

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

   ```
   kubectl describe rolebindingpolicy <name> -n <namespace>
   ```

2.5.10.1.2. Creating a role binding policy from the console

As you create a role binding policy from the console, a YAML file is also created in the YAML editor. Complete the following steps to create a role binding policy from the console:

1. Log in to your cluster from the console.
2. From the navigation menu, click **Governance and risk**.
3. Click **Create policy**.
4. Enter or select the appropriate values for the following fields:
   - Name
   - Specifications
   - Cluster selector
   - Remediation action
   - Standards
   - Categories
   - Controls
   - Disabled
5. Click **Create**.

A role binding policy is created.

2.5.10.1.2.1. Viewing your role binding policy from the console

You can view any role binding policy and its status from the console.

1. Log in to your cluster from the console.

2. From the navigation menu, click **Governance and risk** 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.

3. Select one of your policies to view more details.

4. View the role binding policy violations by selecting the *Status* tab.

2.5.10.2. Updating role binding policies

Learn to update role binding policies by viewing the following section.

2.5.10.2.1. Disabling role binding policies

Complete the following steps to disable your role binding policy:

1. Log in to your Red Hat Advanced Cluster Management for Kubernetes console.

2. From the navigation menu, click **Govern risk** to view a table list of your policies.

3. Disable your policy by clicking the **Actions** icon > **Disable**. The *Disable Policy* dialog box appears.

4. Click **Disable policy**.

Your policy is disabled.

2.5.10.2.2. Deleting a role binding policy

Delete the role binding policy from the CLI or the console.

- Delete a role binding policy from the CLI:

  a. Delete a role binding policy by running the following command:

     ```
     kubectl delete policy <rolebinding-policy-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:

     ```
     kubectl get policy <rolebinding-policy-name> -n <namespace>
     ```

- Delete a role binding policy from the console:

  a. From the navigation menu, click **Govern risk** to view a table list of your policies.

  b. Click the **Actions** icon for the policy you want to delete in the policy violation table.

  c. Click **Remove**.

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

Your role binding policy is deleted.

View a sample of a role binding policy, see *Role binding policy sample* on the Role binding policy page. See Kubernetes configuration policy controller to learn about other configuration policies. See Managing security policies to manage other policies.

2.5.11. Managing Security Context Constraints policies

Learn to create, apply, view, and update your Security Context Constraints (SCC) policies.

2.5.11.1. Creating an SCC policy

You can create a YAML file for your SCC policy from the command line interface (CLI) or from the console. View the following sections to create an SCC policy:

2.5.11.1.1. Creating an SCC policy from the CLI

See Creating Security Context Constraints in the OpenShift Container Platform documentation for more details.

2.5.11.1.1.1. Viewing your SCC policy from the CLI

See Examining an SCC in the OpenShift Container Platform documentation for more details.

2.5.11.1.2. Creating an SCC policy from the console

As you create an SCC policy from the console, a YAML file is also created in the YAML editor. Complete the following steps to create an SCC policy from the console:

1. Log in to your cluster from the console.
2. From the navigation menu, click **Governance and risk**.
3. Click **Create policy**.
4. Enter or select the appropriate values for the following fields:
    - Name
    - Specifications
    - Cluster selector
    - Remediation action
    - Standards
    - Categories
    - Controls
    - Disabled
5. Click **Create**.

An SCC policy is created.

2.5.11.1.2.1. Viewing your SCC policy from the console

You can view any SCC policy and its status from the console.

1. Log in to your cluster from the console.
2. From the navigation menu, click **Governance and risk** 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.
3. Select one of your policies to view more details.
4. View the SCC policy violations by selecting the *Status* tab.

2.5.11.2. Updating SCC policies

Learn to update SCC policies by viewing the following sections.

2.5.11.2.1. Disabling SCC policies

Complete the following steps to disable your SCC policy:

1. Log in to your Red Hat Advanced Cluster Management for Kubernetes console.
2. From the navigation menu, click **Govern risk** to view a table list of your policies.
3. Disable your policy by clicking the **Actions** icon > **Disable**. The *Disable Policy* dialog box appears.
4. Click **Disable policy**.

Your policy is disabled.

2.5.11.2.2. Deleting an SCC policy

Delete the SCC policy from the CLI or the console.

See Deleting an SCC in the OpenShift Container Platform documentation to learn more about deleting an SCC policy from the CLI.

- Delete an SCC policy from the console:
  a. From the navigation menu, click **Govern risk** to view a table list of your policies.
  b. Click the **Actions** icon for the policy you want to delete in the policy violation table.
  c. Click **Remove**.
  d. From the *Remove policy* dialog box, click **Remove policy**.

Your SCC policy is deleted.

To view a sample of an SCC policy, see the *Security context constraint policy sample* section of Security Context Constraints policy. See Kubernetes configuration policy controller to learn about other configuration policies. See Managing security policies to manage other policies.

2.5.12. Managing certificate policies

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

2.5.12.1. Creating a certificate policy

You can create a YAML file for your certificate policy from the command line interface (CLI) or from the console. View the following sections to create a certificate policy:

2.5.12.1.1. Creating a certificate policy from the CLI

Complete the following steps to create a certificate policy from the CLI:

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

   ```
   kubectl create -f policy-1.yaml
   ```

2. Apply the policy by running the following command:

   ```
   kubectl apply -f <certificate-policy-file-name>  --namespace=<namespace>
   ```

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

   ```
   kubectl get certificatepolicy --namespace=<namespace>
   ```

Your certificate policy is created.

2.5.12.1.1.1. Viewing your certificate policy from the CLI

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

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

   ```
   kubectl get certificatepolicy <policy-name> -n <namespace> -o yaml
   ```

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

   ```
   kubectl describe certificatepolicy <name> -n <namespace>
   ```

2.5.12.1.2. Creating a certificate policy from the console

As you create a certificate policy from the console, a YAML file is also created in the YAML editor. Complete the following steps to create a certificate policy from the console:

1. Log in to your cluster from the console.
2. From the navigation menu, click **Governance and risk**.
3. Click **Create policy**.
4. Select *CertificatePolicy* for the *Specifications* parameter. Values for the remaining parameters are automatically set when you select the policy. You can edit your values.
5. Click **Create**.

A certificate policy is created.

2.5.12.1.2.1. Viewing your certificate policy from the console

You can view any certificate policy and its status from the console.

1. Log in to your cluster from the console.
2. From the navigation menu, click **Govern risk** 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.
3. Select one of your policies to view more details. The *Details* tab, *Status* tab, and *YAML* tab are displayed.
4. To view the compliance status of your policy, select the *Status* tab. Click the *View history* link to view a list of violation messages.

2.5.12.2. Updating certificate policies

2.5.12.2.1. Bringing your own certificates

You can monitor your own certificates with the certificate policy controller. You must complete one of the following requirements to monitor your own certificates:

- Create a Kubernetes TLS Secret for your certificate.
- Add the label **certificate_key_name** into your Kubernetes secret to monitor your certificates.

Create a Kubernetes TLS secret to monitor your own certificates by running the following command:

```
kubectl -n <namespace> create secret tls <secret name> --cert=<path to certificate>/<certificate
name> --key=<path to key>/<key name>
```

2.5.12.2.2. Adding a label into your Kubernetes secret

Update the **metadata** parameter in your TLS Secret by adding the **certificate_key_name** label. Run the following command to add the **certificate_key_name** label:

```
kubectl label secret my-certificate -n default certificate_key_name=cert
```

Your updated TLS Secret might resemble the following content:

```
apiVersion: policy.open-cluster-management.io/v1
kind: Secret
metadata:
  name: my-certificate
  namespace: default
  labels:
    certificate_key_name: cert
type: Opaque
data:
  cert: <Certificate Data>
  key: <Private Key Data>
```

Note: When you add the label from the console, you must manually add the label into the TLS Secret YAML file.

2.5.12.2.3. Disabling certificate policies

When you create a certificate policy, it is enabled by default. Complete the following steps to disable a certificate policy from the CLI or the console:

- Disable a certificate policy from the console:
    a. Log in to your Red Hat Advanced Cluster Management for Kubernetes console.
    b. From the navigation menu, click **Govern risk** to view a table list of your policies.
    c. Disable your policy by clicking the **Actions** icon > **Disable**. The *Disable Policy* dialog box appears.
    d. Click **Disable policy**.

Your policy is disabled.

2.5.12.2.4. Deleting a certificate policy

Delete the certificate policy from the CLI or the console.

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

    ```
    kubectl delete policy <cert-policy-name> -n <namespace>
    ```

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

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

    ```
    kubectl get policy <policy-name> -n <mcm namespace>
    ```

- Delete a certificate policy from the console:
    a. From the navigation menu, click **Govern risk** to view a table list of your policies.
    b. Click the **Actions** icon for the policy you want to delete in the policy violation table.
    c. Click **Remove**.
    d. From the *Remove policy* dialog box, click **Remove policy**.

Your certificate policy is deleted.

View a sample of a certificate policy, see policy-certificate.yaml. Refer to Certificate policy controller for more details.

For more information about other policy controllers, see Policy controllers. See Managing security policies to manage other policies.

2.5.13. Managing IAM policies

Apply an IAM policy to check the number of cluster administrators that you allow in your managed cluster. Learn to create, apply, view, and update your IAM policies in the following sections.

2.5.13.1. Creating an IAM policy

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

2.5.13.1.1. Creating an IAM policy from the CLI

Complete the following steps to create an IAM policy from the CLI:

1. Create a YAML file with the IAM policy definition. Run the following command:

   ```
   kubectl create -f iam-policy-1.yaml
   ```

   Your IAM policy might resemble the following YAML file:

   ```
   apiVersion: policy.open-cluster-management.io/v1
   kind: IamPolicy
   metadata:
     name: iam-grc-policy
     label:
       category: "System-Integrity"
   spec:
     namespaceSelector:
       include: ["default","kube-*"]
       exclude: ["kube-system"]
     remediationAction: inform
     disabled: false
     maxClusterRoleBindingUsers: 5
   ```

2. Apply the policy by running the following command:

   ```
   kubectl apply -f <iam-policy-file-name>  --namespace=<namespace>
   ```

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

   ```
   kubectl get <iam-policy-file-name> --namespace=<namespace>
   ```

Your IAM policy is created.

2.5.13.1.1.1. Viewing your IAM policy from the CLI

Complete the following steps to view your IAM policy:

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

   ```
   kubectl get iampolicy <policy-name> -n <namespace> -o yaml
   ```

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

   ```
   kubectl describe iampolicy <name> -n <namespace>
   ```

2.5.13.1.2. Creating an IAM policy from the console

As you create your IAM policy from the console, a YAML file is also created in the YAML editor. Complete the following steps to create an IAM policy from the console:

1. Log in to your cluster from the console.
2. From the navigation menu, click **Govern risk**.
3. Click **Create policy**.
4. Select **IamPolicy** from the *Specifications* field. Values for the remaining parameters are set automatically when you select the policy. You can edit your values.
5. Click **Create**.

An IAM policy is created.

2.5.13.1.2.1. Viewing your IAM policy from the console

You can view any IAM policy and its status from the console.

1. Log in to your cluster from the console.
2. From the navigation menu, click **Govern risk** 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.
3. Select one of your policies to view more details.
4. View the IAM policy violations by selecting the *Status* tab.

2.5.13.2. Updating IAM policies

Learn to update IAM policies by viewing the following section.

2.5.13.2.1. Disabling IAM policies

Complete the following steps to disable your IAM policy:

1. Log in to your Red Hat Advanced Cluster Management for Kubernetes console.
2. From the navigation menu, click **Govern risk** to view a table list of your policies.
3. Disable your policy by clicking the **Actions** icon > **Disable**. The *Disable Policy* dialog box appears.
4. Click **Disable policy**.

Your policy is disabled.

2.5.13.2.2. Deleting an IAM policy

Delete a configuration policy from the CLI or the console.

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

```
kubectl delete policy <iam-policy-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:

```
kubectl get policy <iam-policy-name> -n <namespace>
```

- Delete an IAM policy from the console:
  a. From the navigation menu, click **Govern risk** to view a table list of your policies.
  b. Click the **Actions** icon for the policy you want to delete in the policy violation table.
  c. Click **Remove**.
  d. From the *Remove policy* dialog box, click **Remove policy**.

Your policy is deleted.

View the *IAM policy sample* from the IAM policy controller page. See Managing security policies for more topics.

2.5.14. Managing ETCD encryption policies

Apply an encryption policy to detect, or enable encryption of sensitive data in the ETCD data-store. Learn to create, apply, view, and update your encryption policy in the following sections.

2.5.14.1. Creating an encryption policy

You can create a YAML file for your encryption policy from the command line interface (CLI) or from the console. View the following sections to create a encryption policy:

2.5.14.1.1. Creating an encryption policy from the CLI

Complete the following steps to create an encryption policy from the CLI:

1. Create a YAML file for your encryption policy by running the following command:

```
kubectl create -f etcd-encryption-policy-1.yaml
```

2. Apply the policy by running the following command:

```
kubectl apply -f <etcd-encryption-policy-file-name>  --namespace=<namespace>
```

3. List and verify the policies by running the following command:

```
kubectl get etcd-encryption-policy --namespace=<namespace>
```

Your encryption policy is created from the CLI.

2.5.14.1.1.1. Viewing your encryption policy from the CLI

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

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

   ```
   kubectl get etcd-encryption-policy <policy-name> -n <namespace> -o yaml
   ```

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

   ```
   kubectl describe etcd-encryption-policy <name> -n <namespace>
   ```

2.5.14.1.2. Creating an encryption policy from the console

As you create a encryption policy from the console, a YAML file is also created in the YAML editor. Complete the following steps to create the encryption policy from the console:

1. Log in to your Red Hat Advanced Cluster Management for Kubernetes console.
2. From the navigation menu, click **Govern risk**.
3. Click **Create policy**.
4. Select **EtcdEncryption** from the *Specifications* field. Values for the remaining parameters are set automatically when you select the policy. You can edit your values.
5. Click **Create**.

2.5.14.1.2.1. Viewing your encryption policy from the console

You can view any encryption policy and its status from the console.

1. Log in to your cluster from the console.
2. From the navigation menu, click **Govern risk** 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.
3. Select one of your policies to view more details. The *Overview* tab, *Status* tab, and *YAML* tab are displayed.

2.5.14.2. Updating encryption policies

Learn to update encryption policies by viewing the following section.

2.5.14.2.1. Disabling encryption policies

Complete the following steps to disable your encryption policy:

1. Log in to your Red Hat Advanced Cluster Management for Kubernetes console.
2. From the navigation menu, click **Govern risk** to view a table list of your policies.
3. Disable your policy by clicking the **Actions** icon > **Disable**. The *Disable Policy* dialog box appears.
4. Click **Disable policy**.

Your policy is disabled.

2.5.14.2.2. Deleting an encryption policy

Delete the encryption policy from the CLI or the console.

- Delete an encryption policy from the CLI:

  a. Delete an encryption policy by running the following command:

  ```
  kubectl delete policy <podsecurity-policy-name> -n <namespace>
  ```

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

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

  ```
  kubectl get policy <podsecurity-policy-name> -n <namespace>
  ```

- Delete a encryption policy from the console:

  a. From the navigation menu, click **Govern risk** to view a table list of your policies.

  b. Click the **Actions** icon for the policy you want to delete in the policy violation table.

  c. Click **Remove**.

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

Your encryption policy is deleted.

View a sample of an encryption policy, see *ETCD encryption policy sample* on the ETCD encryption policy page. See Kubernetes configuration policy controller to learn about other configuration policies. See Managing security policies to manage other policies.

2.5.15. 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

2.5.15.1. Installing gatekeeper using a gatekeeper operator policy

[illegible]

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 CR to configure gatekeeper. View the Gatekeeper operator CR 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**.

2.5.15.2. Creating a gatekeeper policy from the console

Complete the following steps to install the gatekeeper operator policy from the console:

1. Log in to your cluster.
2. From the navigation menu, click **Govern risk**.
3. Create a policy by selecting **Create policy**.
4. As you complete the form, select **GatekeeperOperator** 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. See **policy-gatekeeper-operator.yaml** to view an the sample.
   Note: Consider that default values can be generated by the operator. See Gatekeeper Helm Chart for an explanation of the optional parameters that can be used for the gatekeeper operator policy.

2.5.15.2.1. Gatekeeper operator CR

```
apiVersion: operator.gatekeeper.sh/v1alpha1
kind: Gatekeeper
metadata:
  name: gatekeeper
spec:
  # Add fields here
  image:
    image: docker.io/openpolicyagent/gatekeeper:v3.2.2
    imagePullPolicy: Always
  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"
```

2.5.15.3. Upgrading gatekeeper and the gatekeeper operator

You can upgrade the versions for gatekeeper and the gatekeeper operator. Complete the following steps:

- 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**.

- Upgrade the gatekeeper version manually by completing the following steps:

  a. Identify the latest version for gatekeeper, see Red Hat Ecosystem Catalog - gatekeeper .

  b. Select the drop-down menu of the *Tag* filter to find the latest static tag. For example, **v3.3.0-1**.

  c. Edit the gatekeeper operator policy and update the **image** tag to use the latest static tag. View the following example of the updated line in the gatekeeper operator policy:

  ```
  image: 'registry.redhat.io/rhacm2/gatekeeper-rhel8:v3.3.0-1'
  ```

  For more information, see How to use Gatekeeper .

2.5.15.4. Updating gatekeeper operator policy

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

2.5.15.4.1. Viewing gatekeeper operator policy from the console

You can view your gatekeeper operator policy and its status from the console.

1. Log in to your cluster from the console.
2. From the navigation menu, click **Govern risk** 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.
3. Select the **policy-gatekeeper-operator** policy to view more details.
4. View the policy violations by selecting the *Status* tab.

2.5.15.4.2. Disabling gatekeeper operator policy

Complete the following steps to disable your gatekeeper operator policy:

1. Log in to your Red Hat Advanced Cluster Management for Kubernetes console.
2. From the navigation menu, click **Govern risk** to view a table list of your policies.
3. Disable your policy by clicking the **Actions** icon > **Disable**. The *Disable Policy* dialog box appears.
4. Click **Disable policy**.

Your policy is disabled.

2.5.15.4.3. 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:

     ```
     kubectl delete policy <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:

     ```
     kubectl get policy <policy-gatekeeper-operator-name> -n <namespace>
     ```

- Delete gatekeeper operator policy from the console:
  a. From the navigation menu, click **Govern risk** to view a table list of your policies.
  b. Click the **Actions** icon for the **policy-gatekeeper-operator** policy to delete in the policy violation table.
  c. Click **Remove**.

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

Your gatekeeper operator policy is deleted.

2.5.15.5. 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 (CR).
   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.

See Integrating gatekeeper constraints and constraint templates for details about gatekeeper. For a list of topics to integrate third-party policies with the product, see Integrate third-party policy controllers.

2.5.16. Managing compliance operator policies

Apply a compliance operator policy to install the Red Hat OpenShift Container Platform compliance operator. Learn to create, update, apply, and view your compliance operator policy in the following sections.

2.5.16.1. Creating a compliance operator policy from the console

As you create a compliance operator policy from the console, a YAML file is also created in the YAML editor. Complete the following steps to create a compliance operator policy from the console:

1. Log in to your hub cluster.
2. From the navigation menu, select **Govern risk**.

3. Click **Create policy**. As you complete the YAML form, select **ComplianceOperator** from the *Specifications* field.
   The following resources are created: compliance operator namespace (**openshift-compliance**), an operator group (**compliance-operator**), and a subscription (**comp-operator-subscription**).

   Note: Enforce is supported. When you set the remediation action to **enforce** the policy installs the compliance operator.

A compliance operator policy is created.

2.5.16.2. Updating a compliance operator policy

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

2.5.16.2.1. Viewing a compliance operator policy from the console

You can view any compliance operator policy and its status from the console.

1. Log in to your cluster from the console.
2. From the navigation menu, click **Govern risk** 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.
3. Select **policy-comp-operator** policy to view more details.
4. View the policy violations by selecting the *Status* tab.

2.5.16.2.2. Disabling a compliance operator policy

Complete the following steps to disable your compliance operator policy:

1. Log in to your Red Hat Advanced Cluster Management for Kubernetes console.
2. From the navigation menu, click **Govern risk** to view a table list of your policies.
3. Disable **policy-comp-operator** by clicking the **Actions** icon > **Disable**. The *Disable Policy* dialog box appears.
4. Click **Disable policy**.

Your policy is disabled.

2.5.16.2.3. Deleting a compliance operator policy

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

- Delete compliance operator policy from the CLI:
  1. Delete compliance operator policy by running the following command:

     ```
     kubectl delete policy <policy-comp-operator-name> -n <namespace>
     ```

     After your policy is deleted, it is removed from your target cluster or clusters.
  2. Verify that your policy is removed by running the following command:

```
kubectl get policy <policy-comp-operator-name> -n <namespace>
```

- Delete compliance operator policy from the console:
    1. From the navigation menu, click **Govern risk** to view a table list of your policies.
    2. Click the **Actions** icon for the **policy-comp-operator** policy to delete in the policy violation table.
    3. Click **Remove**.
    4. From the *Remove policy* dialog box, click **Remove policy**.

Your compliance operator policy is deleted.

For more details about the compliance operator policy, see Compliance operator policy .

2.5.17. Managing E8 scan policies

Apply an E8 scan policy to scan master and worker nodes for compliance with the E8 profiles. Learn to create, update, apply, and view your E8 scan policy in the following sections.

2.5.17.1. Creating an E8 scan policy from the console

As you create an E8 scan policy from the console, a YAML file is also created in the YAML editor. **Note:** The compliance operator must be installed. For more details, see Creating a compliance operator policy from the console.

Complete the following steps to create a E8 scan policy from the console:

1. Log in to your hub cluster.
2. From the navigation menu, select **Govern risk**.
3. Click **Create policy**. Select *Custom specification* from the *Specification* field. Copy and paste the **policy-e8-scan** from the policy-collection repository.
   The following resources are created: **ScanSettingBinding** (**e8**), a **ComplianceSuite** (**compliance-suite-e8**), and a **ComplianceCheckResult** (**compliance-suite-e8-results**).

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

An E8 scan policy is created.

2.5.17.2. Updating an E8 scan policy

Learn to update the E8 scan policy by viewing the following section.

2.5.17.2.1. Viewing an E8 scan policy from the console

You can view any E8 scan policy and its status from the console.

1. Log in to your cluster from the console.
2. From the navigation menu, click **Govern risk** 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.

3. Select **policy-compliance-operator-e8-scan** policy to view more details.

4. View the policy violations by selecting the *Status* tab.

2.5.17.2.2. Disabling an E8 scan policy

Complete the following steps to disable your compliance operator policy:

1. Log in to your Red Hat Advanced Cluster Management for Kubernetes console.

2. From the navigation menu, click **Govern risk** to view a table list of your policies.

3. Disable **policy-compliance-operator-e8-scan** by clicking the **Actions** icon > **Disable**. The *Disable Policy* dialog box appears.

4. Click **Disable policy**.

Your policy is disabled.

2.5.17.2.3. Deleting an E8 scan policy

Delete the E8 scan policy from the CLI or the console.

- Delete an E8 scan policy from the CLI:

  1. Delete an E8 policy by running the following command:

     ```
     kubectl delete policy <policy-compliance-operator-e8-scan> -n <namespace>
     ```

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

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

     ```
     kubectl get policy <policy-compliance-operator-e8-scan> -n <namespace>
     ```

- Delete an E8 scan policy from the console:

  1. From the navigation menu, click **Govern risk** to view a table list of your policies.

  2. Click the **Actions** icon for the **policy-compliance-operator-e8-scan** policy to delete in the policy violation table.

  3. Click **Remove**.

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

Your E8 scan policy is deleted.

For more details about the E8 scan policy, see E8 scan policy.