Red Hat Advanced Cluster Management for Kubernetes 2.10

GitOps

GitOps

Last Updated: 2024-06-12

Red Hat Advanced Cluster Management for Kubernetes 2.10 GitOps

GitOps

Legal Notice

Abstract

Read more to learn how to use integrated GitOps and Argo CD.

Table of Contents

CHAPTER 1. GITOPS OVERVIEW

Red Hat OpenShift Container Platform GitOps and Argo CD is integrated with Red Hat Advanced Cluster Management for Kubernetes, with advanced features compared to the original Application Lifecycle *Channel* and *Subscription* model.

GitOps integration with Argo CD development is active, as well as the large community that contributes feature enhancements and updates to Argo CD. By utilizing the OpenShift Container Platform GitOps Operator, you can use the latest advancements in Argo CD development and receive support from the GitOps Operator subscription.

See the following topics to learn more about Red Hat Advanced Cluster Management for Kubernetes integration with OpenShift Container Platform GitOps and Argo CD:

- GitOps console
- Registering managed clusters to OpenShift GitOps operator
- Configuring application placement tolerations for GitOps
- Deploying Argo CD with the Push and Pull model
- Generating a policy to install GitOps Operator
- Managing policy definitions with OpenShift Container Platform GitOps (Argo CD)

1.1. GITOPS CONSOLE

Learn more about integrated OpenShift Container Platform GitOps console features. Create and view applications, such as *ApplicationSet*, and *Argo CD* types. An **ApplicationSet** represents Argo applications that are generated from the controller.

- For an Argo CD **ApplicationSet** to be created, you need to enable **Automatically sync when cluster state changes** from the **Sync policy**.
- For Flux with the **kustomization** controller, find Kubernetes resources with the label **kustomize.toolkit.fluxcd.io/name=<app_name>**.
- For Flux with the **helm** controller, find Kubernetes resources with the label **helm.toolkit.fluxcd.io/name=<app_name>**.
- You need GitOps cluster resources and the GitOps operator installed to create an **ApplicationSet**. Without these prerequisites, you will see no **Argo server** options in the console to create an **ApplicationSet**.

Important: Available actions are based on your assigned role. Learn about access requirements from the Role-based access control documentation.

- Click **Launch resource in Search** to search for related resources.
- Use *Search* to find application resources by the component **kind** for each resource. To search for resources, use the following values:

1.1.1. Querying Argo CD applications

When you search for an Argo CD application, you are directed to the *Applications* page. Complete the following steps to access the Argo CD application from the *Search* page:

1. Log in to your Red Hat Advanced Cluster Management hub cluster.
2. From the console header, select the *Search* icon.
3. Filter your query with the following values: **kind:application** and **apigroup:argoproj.io**.
4. Select an application to view. The *Application* page displays an overview of information for the application.

For more information about search, see Searching in the console introduction .

1.2. REGISTERING MANAGED CLUSTERS TO OPENSHIFT GITOPS OPERATOR

To configure GitOps with the Push model, you can register a set of one or more Red Hat Advanced Cluster Management for Kubernetes managed clusters to an instance of Red Hat OpenShift Container Platform GitOps operator. After registering, you can deploy applications to those clusters. Set up a continuous GitOps environment to automate application consistency across clusters in development, staging, and production environments.

1.2.1. Prerequisites

1. You need to install the Red Hat OpenShift GitOps operator on your Red Hat Advanced Cluster Management for Kubernetes.
2. Import one or more managed clusters.

1.2.2. Registering managed clusters to GitOps

Complete the following steps to register managed clusters to GitOps:

1. Create managed cluster set bindings and add managed clusters to those managed cluster set bind. See the example for managed cluster sets in the **multicloud-integrations managedclusterset**.
   See the Creating a *ManagedClusterSet* documentation for more information.
2. Create a managed cluster set binding to the namespace where Red Hat OpenShift GitOps is deployed. For an example of binding the managed cluster to the **openshift-gitops** namespace, see the **multicloud-integrations** managed clusterset binding example. In the *Additional resources* section, see *Creating a ManagedClusterSetBinding resource* for more general information about creating a **ManagedClusterSetBinding**. See Filtering ManagedClusters from ManagedClusterSets for placement information.
3. In the namespace that is used in managed cluster set binding, create a **Placement** custom resource to select a set of managed clusters to register to an OpenShift Container Platform GitOps operator instance. Use the **multicloud-integration** placement example as a template. See *Using ManagedClusterSets with Placement* for placement information.
   Notes:
   - Only OpenShift Container Platform clusters are registered to an Red Hat OpenShift Container Platform GitOps operator instance, not other Kubernetes clusters.

- In some unstable network scenarios, the managed clusters might be temporarily placed in **unavailable** or **unreachable** state. See *Configuring placement tolerations for Red Hat Advanced Cluster Management and OpenShift GitOps* for more details.

4. Create a **GitOpsCluster** custom resource to register the set of managed clusters from the placement decision to the specified instance of OpenShift GitOps. This enables the OpenShift GitOps instance to deploy applications to any of those Red Hat Advanced Cluster Management managed clusters. Use the ***multicloud-integrations*** GitOps cluster example.
Note: The referenced **Placement** resource must be in the same namespace as the **GitOpsCluster** resource. See the following example:

```
apiVersion: apps.open-cluster-management.io/v1beta1
kind: GitOpsCluster
metadata:
  name: gitops-cluster-sample
  namespace: dev
spec:
  argoServer:
    cluster: local-cluster
    argoNamespace: openshift-gitops
  placementRef:
    kind: Placement
    apiVersion: cluster.open-cluster-management.io/v1beta1
    name: all-openshift-clusters 1
```

1 The **placementRef.name** value is **all-openshift-clusters**, and is specified as target clusters for the GitOps instance that is installed in **argoNamespace: openshift-gitops**. The **argoServer.cluster** specification requires the **local-cluster** value.

5. Save your changes. You can now follow the GitOps workflow to manage your applications.

1.2.3. GitOps token

When you integrate with the GitOps operator for every managed cluster that is bound to the GitOps namespace through the placement and **ManagedClusterSetBinding** custom resources, a secret with a token to access the **ManagedCluster** is created in the namespace. This is required for the GitOps controller to sync resources to the managed cluster. When a user is given administrator access to a GitOps namespace to perform application lifecycle operations, the user also gains access to this secret and **admin** level to the managed cluster.

If this is not desired, instead of binding the user to the namespace-scoped **admin** role, use a more restrictive custom role with permissions required to work with application resources that can be created and used to bound the user. See the following **ClusterRole** example:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: application-set-admin
rules:
- apiGroups:
  - argoproj.io
  resources:
  - applicationsets
  verbs:
```

```
- get
- list
- watch
- update
- delete
- deletecollection
- patch
```

1.2.4. Additional resources

- Refer to Configuring application placement tolerations for GitOps for more details.
- See the **multicloud-integrations** managed cluster set example.
- Refer to Creating a *ManagedClusterSet*
- See the **multicloud-integration** placement example.
- See Placement overview for placement information.
- See the **multicloud-integrations** GitOps cluster example.
- See the **multicloud-integrations** managed cluster set binding example.
- See Creating a *ManagedClusterSetBinding* resource documentation for more information.
- See About GitOps to learn more.

1.3. CONFIGURING APPLICATION PLACEMENT TOLERATIONS FOR GITOPS

Red Hat Advanced Cluster Management provides a way for you to register managed clusters that deploy applications to Red Hat OpenShift GitOps.

In some unstable network scenarios, the managed clusters might be temporarily placed in **Unavailable** state. If a **Placement** resource is being used to facilitate the deployment of applications, add the following tolerations for the **Placement** resource to continue to include unavailable clusters. The following example shows a **Placement** resource with tolerations:

```
apiVersion: cluster.open-cluster-management.io/v1beta1
kind: Placement
metadata:
  name: placement
  namespace: ns1
spec:
  tolerations:
    - key: cluster.open-cluster-management.io/unreachable
      operator: Exists
    - key: cluster.open-cluster-management.io/unavailable
      operator: Exists
```

1.4. DEPLOYING ARGO CD WITH PUSH AND PULL MODEL

Using a *Push model*, The Argo CD server on the hub cluster deploys the application resources on the managed clusters. For the *Pull model*, the application resources are propagated by the *Propagation controller* to the managed clusters by using **manifestWork**.

For both models, the same **ApplicationSet** CRD is used to deploy the application to the managed cluster.

Required access: Cluster administrator

- Prerequisites
- Architecture
- Creating the *ApplicationSet* custom resource
- *MulticlusterApplicationSetReport*

1.4.1. Prerequisites

View the following prerequisites for the Argo CD Pull model:

Important:

- If your **openshift-gitops-ArgoCD-application-controller** service account is *not* assigned as a cluster administrator, the GitOps application controller might not deploy resources. The application status might send an error similar to the following error:

```
cannot create resource "services" in API group "" in the namespace
"mortgage",deployments.apps is forbidden: User
"system:serviceaccount:openshift-gitops:openshift-gitops-Argo CD-application-controller"
```

- After you install the **OpenShift Gitops** operator on the managed clusters, you must create the **ClusterRoleBinding** cluster administrator privileges on the same managed clusters.
- To add the **ClusterRoleBinding** cluster administrator privileges to your managed clusters, see the following example YAML:

```
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: argo-admin
subjects:
  - kind: ServiceAccount
    name: openshift-gitops-argocd-application-controller
    namespace: openshift-gitops
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
```

- If you are not a cluster administrator and need to resolve this issue, complete the following steps:

  1. Create all namespaces on each managed cluster where the Argo CD application will be deployed.

2. Add the **managed-by** label to each namespace. If an Argo CD application is deployed to multiple namespaces, each namespace should be managed by Argo CD.
   See the following example with the **managed-by** label:

```
apiVersion: v1
kind: Namespace
metadata:
  name: mortgage2
  labels:
    argocd.argoproj.io/managed-by: openshift-gitops
```

1. You must declare all application destination namespaces in the repository for the application and include the **managed-by** label in the namespaces. Refer to *Additional resources* to learn how to declare a namespace.

See the following requirements to use the Argo CD *Pull* model:

- The GitOps operator must be installed on the hub cluster and the target managed clusters in the **openshift-gitops** namespace.
- The required hub cluster OpenShift Container Platform GitOps operator must be version 1.9.0 or later.
- The required managed clusters OpenShift Container Platform GitOps operator must be the same version as the hub cluster.
- You need the *ApplicationSet controller* to propagate the Argo CD application template for a managed cluster.
- Every managed cluster must have a cluster secret in the Argo CD server namespace on the hub cluster, which is required by the ArgoCD application set controller to propagate the Argo CD application template for a managed cluster.
  To create the cluster secret, create a **gitOpsCluster** resource that contains a reference to a **placement** resource. The **placement** resource selects all the managed clusters that need to support the Pull model. When the GitOps cluster controller reconciles, it creates the cluster secrets for the managed clusters in the Argo CD server namespace.

1.4.2. Architecture

For both Push and Pull model, the *Argo CD ApplicationSet controller* on the hub cluster reconciles to create application resources for each target managed cluster. See the following information about architecture for both models:

1.4.2.1. Architecture Push model

- With Push model, OpenShift Container Platform GitOps applies resources *directly* from a centralized hub cluster to the managed clusters.
- An Argo CD application that is running on the hub cluster communicates with the GitHub repository and deploys the manifests directly to the managed clusters.
- Push model implementation only contains the Argo CD application on the hub cluster, which has credentials for managed clusters. The Argo CD application on the hub cluster can deploy the applications to the managed clusters.

- **Important:** With a large number of managed clusters that require resource application, consider potential strain on the OpenShift Container Platform GitOps controller memory and CPU usage. To optimize resource management, refer to Configuring resource quota or requests .

- By default, the Push model is used to deploy the application unless you add the **apps.open-cluster-management.io/ocm-managed-cluster** and **apps.open-cluster-management.io/pull-to-ocm-managed-cluster** annotations to the template section of the **ApplicationSet**.

1.4.2.2. Architecture Pull model

- Pull model can provide scalability relief compared to the push model by reducing stress on the controller in the hub cluster, but with more requests and status reporting required.

- With Pull model, OpenShift Container Platform GitOps *does not* apply resources directly from a centralized hub cluster to the managed clusters. The Argo CD Application is propagated from the hub cluster to the managed clusters.

- Pull model implementation applies OpenShift Cluster Manager registration, placement, and **manifestWork** APIs so that the hub cluster can use the secure communication channel between the hub cluster and the managed cluster to deploy resources.

- Each managed cluster individually communicates with the GitHub repository to deploy the resource manifests locally, so you must install and configure GitOps operators on each managed cluster.

- An Argo CD server must be running on each target managed cluster. The Argo CD application resources are replicated on the managed clusters, which are then deployed by the local Argo CD server. The distributed Argo CD applications on the managed clusters are created with a single Argo CD **ApplicationSet** resource on the hub cluster.

- The managed cluster is determined by the value of the **ocm-managed-cluster** annotation.

- For successful implementation of Pull model, the Argo CD application controller must *ignore* Push model application resources with the **argocd.argoproj.io/skip-reconcile** annotation in the template section of the **ApplicationSet**.

- For Pull model, the *Argo CD Application controller* on the managed cluster reconciles to deploy the application.

- The Pull model *Resource sync controller* on the hub cluster queries the OpenShift Cluster Manager search V2 component on each managed cluster periodically to retrieve the resource list and error messages for each Argo CD application.

- The *Aggregation controller* on the hub cluster creates and updates the **MulticlusterApplicationSetReport** from across clusters by using the data from the Resource sync controller, and the status information from **manifestWork**.

- The status of the deployments is gathered back to the hub cluster, but not all the detailed information is transmitted. Additional status updates are periodically scraped to provide an overview. The status feedback is not real-time, and each managed cluster GitOps operator needs to communicate with the Git repository, which results in multiple requests.

1.4.3. Creating the *ApplicationSet* custom resource

The Argo CD **ApplicationSet** resource is used to deploy applications on the managed clusters by using the Push or Pull model with a **placement** resource in the generator field that is used to get a list of managed clusters.

1. For the Pull model, set the destination for the application to the default local Kubernetes server, as displayed in the following example. The application is deployed locally by the application controller on the managed cluster.

2. Add the annotations that are required to override the default Push model, as displayed in the following example **ApplicationSet** YAML, which uses the Pull model with template annotations:

```
apiVersion: argoproj.io/v1alpha1
kind: `ApplicationSet`
metadata:
  name: guestbook-allclusters-app-set
  namespace: openshift-gitops
spec:
  generators:
  - clusterDecisionResource:
      configMapRef: ocm-placement-generator
      labelSelector:
        matchLabels:
          cluster.open-cluster-management.io/placement: aws-app-placement
      requeueAfterSeconds: 30
  template:
    metadata:
      annotations:
        apps.open-cluster-management.io/ocm-managed-cluster: '{{name}}' 1
        apps.open-cluster-management.io/ocm-managed-cluster-app-namespace: openshift-
gitops
        argocd.argoproj.io/skip-reconcile: "true" 2
      labels:
        apps.open-cluster-management.io/pull-to-ocm-managed-cluster: "true" 3
      name: '{{name}}-guestbook-app'
    spec:
      destination:
        namespace: guestbook
        server: https://kubernetes.default.svc
      project: default
      sources: [
      {
        repoURL: https://github.com/argoproj/argocd-example-apps.git
        targetRevision: main
        path: guestbook
         }
      ]
      syncPolicy:
        automated: {}
        syncOptions:
        - CreateNamespace=true
```

1 The **apps.open-cluster-management.io/ocm-managed-cluster** is needed for the Pull model.

2 The **argocd.argoproj.io/skip-reconcile** is needed to ignore the Push model resources.

3 The **apps.open-cluster-management.io/pull-to-ocm-managed-cluster: "true"** is also needed for the Pull model.

1.4.4. *MulticlusterApplicationSetReport*

- For the Pull model, the **MulticlusterApplicationSetReport** aggregates application status from across your managed clusters.
- The report includes the list of resources and the overall status of the application from each managed cluster.
- A separate report resource is created for each Argo CD ApplicationSet resource. The report is created in the same namespace as the **ApplicationSet**.
- The report includes the following items:
  1. A list of resources for the Argo CD application
  2. The overall sync and health status for each Argo CD application
  3. An error message for each cluster where the overall status is **out of sync** or **unhealthy**
  4. A summary status all the states of your managed clusters
- The *Resource sync controller* and the *Aggregation controller* both run every 10 seconds to create the report.
- The two controllers, along with the Propagation controller, run in separate containers in the same **multicluster-integrations** pod, as shown in the following example output:

```
NAMESPACE              NAME                                        READY   STATUS
open-cluster-management multicluster-integrations-7c46498d9-fqbq4  3/3     Running
```

The following is an example **MulticlusterApplicationSetReport** YAML file for the **guestbook** application:

```
apiVersion: apps.open-cluster-management.io/v1alpha1
kind: MulticlusterApplicationSetReport
metadata:
  labels:
    apps.open-cluster-management.io/hosting-applicationset: openshift-gitops.guestbook-allclusters-app-set
  name: guestbook-allclusters-app-set
  namespace: openshift-gitops
statuses:
  clusterConditions:
  - cluster: cluster1
    conditions:
    - message: 'Failed sync attempt: one or more objects failed to apply, reason: services is forbidden: User "system:serviceaccount:openshift-gitops:openshift-gitops-Argo CD-application-controller" cannot create resource "services" in API group "" in the namespace "guestbook",deployments.apps is forbidden: User <name> cannot create resource "deployments" in API group "apps" in the namespace "guestboo...'
      type: SyncError
    healthStatus: Missing
```

```
    syncStatus: OutOfSync
  - cluster: pcluster1
    healthStatus: Progressing
    syncStatus: Synced
  - cluster: pcluster2
    healthStatus: Progressing
    syncStatus: Synced
  summary:
    clusters: "3"
    healthy: "0"
    inProgress: "2"
    notHealthy: "3"
    notSynced: "1"
    synced: "2"
```

Note: If a resource fails to deploy, the resource is not included in the resource list. See error messages for information.

1.4.5. Additional resources

- See Configuring an OpenShift cluster by deploying an application with cluster configurations in the OpenShift Container Platform documentation.
- See Setting up an Argo CD instance in the OpenShift Container Platform documentation.

1.5. MANAGING POLICY DEFINITIONS WITH OPENSHIFT CONTAINER PLATFORM GITOPS (ARGO CD)

Deprecated: PlacementRule

Based on Argo CD, you can use OpenShift Container Platform GitOps to manage policy definitions. To allow this workflow, you must grant OpenShift Container Platform GitOps access for you to create policies on the Red Hat Advanced Cluster Management hub cluster. Complete the following steps to create a **ClusterRole** resource for OpenShift Container Platform GitOps, with access to create, read, update, and delete policies and placements:

1. Create a **ClusterRole** from the console. Your **ClusterRole** might resemble the following example:

```
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: openshift-gitops-policy-admin
rules:
  - verbs:
      - get
      - list
      - watch
      - create
      - update
      - patch
      - delete
    apiGroups:
      - policy.open-cluster-management.io
    resources:
```

```
      - policies
      - policysets
      - placementbindings
    - verbs:
      - get
      - list
      - watch
      - create
      - update
      - patch
      - delete
    apiGroups:
      - apps.open-cluster-management.io
    resources:
      - placementrules
    - verbs:
      - get
      - list
      - watch
      - create
      - update
      - patch
      - delete
    apiGroups:
      - cluster.open-cluster-management.io
    resources:
      - placements
      - placements/status
      - placementdecisions
      - placementdecisions/status
```

2. Create a **ClusterRoleBinding** object to grant the OpenShift Container Platform GitOps service account access to the **openshift-gitops-policy-admin ClusterRole** object. Your **ClusterRoleBinding** might resemble the following example:

```
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: openshift-gitops-policy-admin
subjects:
  - kind: ServiceAccount
    name: openshift-gitops-argocd-application-controller
    namespace: openshift-gitops
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: openshift-gitops-policy-admin
```

When a Red Hat Advanced Cluster Management policy definition is deployed with OpenShift Container Platform GitOps, a copy of the policy is created in each managed cluster namespace. These copies are called replicated policies. To prevent OpenShift Container Platform GitOps from repeatedly deleting this replicated policy or show that the ArgoCD **Application** is out of sync, the **argocd.argoproj.io/compare-options: IgnoreExtraneous** annotation is automatically set on each replicated policy by the Red Hat Advanced Cluster Management policy framework.

There are labels and annotations used by Argo CD to track objects. For replicated policies to not show up at all in Argo CD, you can set **spec.copyPolicyMetadata** to **false** on the Red Hat Advanced Cluster Management policy definition to disable the Argo CD tracking labels and annotations from being copied to the replicated policy.

1.5.1. Integrating the Policy Generator with OpenShift Container Platform GitOps (Argo CD)

Based on Argo CD, you can use OpenShift Container Platform GitOps to generate policies by using the Policy Generator through GitOps. Since the Policy Generator does not come preinstalled in the OpenShift Container Platform GitOps container image, some customization must take place. In order to follow along, you must have the OpenShift Container Platform GitOps Operator installed on the Red Hat Advanced Cluster Management hub cluster and be sure to log in to the hub cluster.

For OpenShift Container Platform GitOps to have access to the Policy Generator when you run Kustomize, an Init Container is required to copy the Policy Generator binary from the Red Hat Advanced Cluster Management Application Subscription container image to the OpenShift Container Platform GitOps container. Additionally, OpenShift Container Platform GitOps must be configured to provide the **--enable-alpha-plugins** flag when it runs Kustomize. Complete the following steps:

1. Start editing the OpenShift Container Platform GitOps **argocd** object with the following command:

```
oc -n openshift-gitops edit argocd openshift-gitops
```

2. Modify the OpenShift Container Platform GitOps **argocd** object to contain the following additional YAML content. When a new major version of Red Hat Advanced Cluster Management is released and you want to update the Policy Generator to a newer version, you need to update the **registry.redhat.io/rhacm2/multicluster-operators-subscription-rhel8** image used by the Init Container to a newer tag. View the following example and replace **<version>** with 2.10 or your desired Red Hat Advanced Cluster Management version:

```
apiVersion: argoproj.io/v1alpha1
kind: ArgoCD
metadata:
  name: openshift-gitops
  namespace: openshift-gitops
spec:
  kustomizeBuildOptions: --enable-alpha-plugins
  repo:
    env:
    - name: KUSTOMIZE_PLUGIN_HOME
      value: /etc/kustomize/plugin
    initContainers:
    - args:
      - -c
      - cp /policy-generator/PolicyGenerator-not-fips-compliant /policy-generator-
tmp/PolicyGenerator
      command:
      - /bin/bash
      image: registry.redhat.io/rhacm2/multicluster-operators-subscription-rhel9:v<version>
      name: policy-generator-install
      volumeMounts:
      - mountPath: /policy-generator
        name: policy-generator-tmp
```

```
    volumeMounts:
    - mountPath: /etc/kustomize/plugin/policy.open-cluster-management.io/v1/policygenerator
      name: policy-generator
    volumes:
    - emptyDir: {}
      name: policy-generator
```

Note: Alternatively, you can create a **ConfigurationPolicy** resource that contains the **ArgoCD** manifest and template the version to match the version set in the **MulticlusterHub**:

```
image: '{{ (index (lookup "apps/v1" "Deployment" "open-cluster-management" "multicluster-operators-hub-subscription").spec.template.spec.containers 0).image }}'
```

If you want to enable the processing of Helm charts inside of a Kustomize directory before generating policies, set the environment variable **POLICY_GEN_ENABLE_HELM** to **"true"** in the **spec.repo.env** field:

```
env:
- name: POLICY_GEN_ENABLE_HELM
  value: "true"
```

3. Now that OpenShift Container Platform GitOps can use the Policy Generator, OpenShift Container Platform GitOps must be granted access to create policies on the Red Hat Advanced Cluster Management hub cluster. Create the **ClusterRole** resource called **openshift-gitops-policy-admin**, with access to create, read, update, and delete policies and placements. Refer to the ealier **ClusterRole** resource example.

4. Create a **ClusterRoleBinding** object to grant the OpenShift Container Platform GitOps service account access to the **openshift-gitops-policy-admin ClusterRole**. Your **ClusterRoleBinding** might resemble the following resource:

```
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: openshift-gitops-policy-admin
subjects:
  - kind: ServiceAccount
    name: openshift-gitops-argocd-application-controller
    namespace: openshift-gitops
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: openshift-gitops-policy-admin
```

1.5.2. Additional resources

- Refer to Argo CD documentation.

1.6. GENERATING A POLICY TO INSTALL GITOPS OPERATOR

A common use of Red Hat Advanced Cluster Management policies is to install an Operator on one or more managed Red Hat OpenShift Container Platform clusters. Continue reading to learn how to generate policies by using the Policy Generator, and to install the OpenShift Container Platform GitOps Operator with a generated policy:

1.6.1. Generating a policy that installs OpenShift Container Platform GitOps

You can generate a policy that installs OpenShift Container Platform GitOps by using the Policy Generator. The OpenShift Container Platform GitOps operator offers the *all namespaces* installation mode, which you can view in the following example. Create a **Subscription** manifest file called **openshift-gitops-subscription.yaml**, similar to the following example:

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: openshift-gitops-operator
  namespace: openshift-operators
spec:
  channel: stable
  name: openshift-gitops-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
```

To pin to a specific version of the operator, add the following parameter and value: **spec.startingCSV: openshift-gitops-operator.v<version>**. Replace **<version>** with your preferred version.

A **PolicyGenerator** configuration file is required. Use the configuration file named **policy-generator-config.yaml** to generate a policy to install OpenShift Container Platform GitOps on all OpenShift Container Platform managed clusters. See the following example:

```
apiVersion: policy.open-cluster-management.io/v1
kind: PolicyGenerator
metadata:
  name: install-openshift-gitops
policyDefaults:
  namespace: policies
  placement:
    clusterSelectors:
      vendor: "OpenShift"
  remediationAction: enforce
policies:
  - name: install-openshift-gitops
    manifests:
      - path: openshift-gitops-subscription.yaml
```

The last required file is **kustomization.yaml**, which requires the following configuration:

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

The generated policy might resemble the following file with **PlacementRule**(Deprecated):

```
apiVersion: apps.open-cluster-management.io/v1
kind: PlacementRule
metadata:
  name: placement-install-openshift-gitops
  namespace: policies
spec:
  clusterConditions:
    - status: "True"
```

```
      type: ManagedClusterConditionAvailable
  clusterSelector:
    matchExpressions:
      - key: vendor
        operator: In
        values:
          - OpenShift
---
apiVersion: policy.open-cluster-management.io/v1
kind: PlacementBinding
metadata:
  name: binding-install-openshift-gitops
  namespace: policies
placementRef:
  apiGroup: apps.open-cluster-management.io
  kind: PlacementRule
  name: placement-install-openshift-gitops
subjects:
  - apiGroup: policy.open-cluster-management.io
    kind: Policy
    name: install-openshift-gitops
---
apiVersion: policy.open-cluster-management.io/v1
kind: Policy
metadata:
  annotations:
    policy.open-cluster-management.io/categories: CM Configuration Management
    policy.open-cluster-management.io/controls: CM-2 Baseline Configuration
    policy.open-cluster-management.io/standards: NIST SP 800-53
    policy.open-cluster-management.io/description:
  name: install-openshift-gitops
  namespace: policies
spec:
  disabled: false
  policy-templates:
    - objectDefinition:
        apiVersion: policy.open-cluster-management.io/v1
        kind: ConfigurationPolicy
        metadata:
          name: install-openshift-gitops
        spec:
          object-templates:
            - complianceType: musthave
              objectDefinition:
                apiVersion: operators.coreos.com/v1alpha1
                kind: Subscription
                metadata:
                  name: openshift-gitops-operator
                  namespace: openshift-operators
                spec:
                  channel: stable
                  name: openshift-gitops-operator
                  source: redhat-operators
                  sourceNamespace: openshift-marketplace
          remediationAction: enforce
          severity: low
```

Generated policies from manifests in the OpenShift Container Platform documentation is supported. Any configuration guidance from the OpenShift Container Platform documentation can be applied using the Policy Generator.

1.6.2. Using policy dependencies with *OperatorGroups*

When you install an operator with an **OperatorGroup** manifest, the **OperatorGroup** must exist on the cluster before the **Subscription** is created. Use the policy dependency feature along with the Policy Generator to ensure that the **OperatorGroup** policy is compliant before you enforce the **Subscription** policy.

Set up policy dependencies by listing the manifests in the order that you want. For example, you might want to create the namespace policy first, create the **OperatorGroup** next, and create the **Subscription** last.

Enable the **policyDefaults.orderManifests** parameter and disable **policyDefaults.consolidateManifests** in the Policy Generator configuration manifest to automatically set up dependencies between the manifests.

1.6.3. Additional resources

- See Generating a policy that installs the Compliance Operator .
- See Deploying policies by using GitOps for more details.
- See Understanding OpenShift GitOps and the Operator documentation for more details.
- See Adding Operators to a cluster - Installing from OperatorHub using the CLI
- See the Compliance Operator documentation for more details.
- See *all namespaces* installation mode.
- See *namespaced* installation mode.
- See Using Init Containers to perform tasks before a pod is deployed .
- See Argo CD.
- View the following examples of YAML input that OpenShift Container Platform supports:
  - Post-installation cluster tasks
  - Configuring the audit log policy
  - About forwarding logs to third-party systems

1.7. CREATING A CUSTOMIZED SERVICE ACCOUNT FOR ARGO CD PUSH MODEL

Create a service account on a managed cluster by creating a **managedserviceaccount** resource on the hub cluster. Use the **clusterpermission** resource to grant specific permissions to the service account.

Creating a customzied service account for use with the Argo CD push model includes the following benefits:

- An Application manager add-on runs on each managed cluster. By default, the Argo CD controller uses the service account Application manager to push these resources to the managed cluster.
- The Application manager service account has a large set of permissions because the application subscription add-on uses the Application manager service to deploy applications on the managed cluster. Do not use the Application manager service account if you want a limited set of permissions.
- You can specify a different service account that you want the Argo CD push model to use. When the Argo CD controller pushes resources from the centralized hub cluster to the managed cluster, you can use a different service account than the default Application manager. By using a different service account, you can control the permissions that are granted to this service account.
- The service account must exist on the managed cluster. To facilitate the creation of the service account with the associated permissions, use the **managedserviceaccount** resource and the new **clusterpermission** resource on the centralized hub cluster.

After completing all the following procedures, you can grant cluster permissions to your managed service account. Having the cluster permissions, your managed service account has the necessary permissions to deploy your application resources on the managed clusters. Complete the following procedures:

1. Section 1.7.1, "Creating a managed service account"
2. Section 1.7.2, "Creating a cluster permission"
3. Section 1.7.3, "Using a managed service account in the GitOpsCluster resource"
4. Section 1.7.4, "Creating an Argo CD application"
5. Section 1.7.5, "Using policy to create managed service accounts and cluster permissions"

1.7.1. Creating a managed service account

The **managedserviceaccount** custom resource on the hub provides a convenient way to create **serviceaccounts** on the managed clusters. When a **managedserviceccount** custom resource is created in the **<managed_cluster>** namespace on the hub cluster, a **serviceccount** is created on the managed cluster.

To create a managed service account, see Enabling managedserviceaccount add-ons.

1.7.2. Creating a cluster permission

When the service account is created, it does not have any permission associated to it. To grant permissions to the new service account, use the **clusterpermission** resource. The **clusterpermission** resource is created in the managed cluster namespace on the hub. It provides a convenient way to create roles, cluster roles resources on the managed clusters, and bind them to a service account through a **rolebinding** or **clusterrolebinding** resource.

1. To grant the **<managed-sa-sample>** service account permission to the sample mortgage application that is deployed to the mortgage namespace on **<managed cluster>**, create a YAML with the following content:

```
apiVersion: rbac.open-cluster-management.io/v1alpha1
```

```
kind: ClusterPermission
metadata:
  name: <clusterpermission-msa-subject-sample>
  namespace: <managed cluster>
spec:
  roles:
  - namespace: default
    rules:
    - apiGroups: ["apps"]
      resources: ["deployments"]
      verbs: ["get", "list", "create", "update", "delete", "patch"]
    - apiGroups: [""]
      resources: ["configmaps", "secrets", "pods", "podtemplates", "persistentvolumeclaims",
"persistentvolumes"]
      verbs: ["get", "update", "list", "create", "delete", "patch"]
    - apiGroups: ["storage.k8s.io"]
      resources: ["*"]
      verbs: ["list"]
  - namespace: mortgage
    rules:
    - apiGroups: ["apps"]
      resources: ["deployments"]
      verbs: ["get", "list", "create", "update", "delete", "patch"]
    - apiGroups: [""]
      resources: ["configmaps", "secrets", "pods", "services", "namespace"]
      verbs: ["get", "update", "list", "create", "delete", "patch"]
  clusterRole:
    rules:
    - apiGroups: ["*"]
      resources: ["*"]
      verbs: ["get", "list"]
  roleBindings:
  - namespace: default
    roleRef:
      kind: Role
    subject:
      apiGroup: authentication.open-cluster-management.io
      kind: ManagedServiceAccount
      name: <managed-sa-sample>
  - namespace: mortgage
    roleRef:
      kind: Role
    subject:
      apiGroup: authentication.open-cluster-management.io
      kind: ManagedServiceAccount
      name: <managed-sa-sample>
  clusterRoleBinding:
    subject:
      apiGroup: authentication.open-cluster-management.io
      kind: ManagedServiceAccount
      name: <managed-sa-sample>
```

2. Save the YAML file in a file called, **cluster-permission.yaml**.

3. Run **oc apply -f cluster-permission.yaml**.

4. The sample **<clusterpermission>** creates a role called **<clusterpermission-msa-subject-sample>** in the mortgage namespace. If one does not already exist, create a namespace **mortgage**.

5. Review the resources that are created on the **<managed cluster>**.

After you create the sample, **<clusterpermission>**, the following resources are created in the sample managed cluster:

- One role called **<clusterpermission-msa-subject-sample>** in the default namespace.
- One roleBinding called **<clusterpermission-msa-subject-sample>** in the default namespace for binding the role to the managed service account.
- One role called **<clusterpermission-msa-subject-sample>** in the mortgage namespace.
- One roleBinding called **<clusterpermission-msa-subject-sample>** in the mortgage namespace for binding the role to the managed service account.
- One clusterRole called **<clusterpermission-msa-subject-sample>**.
- One clusterRoleBinding called **<clusterpermission-msa-subject-sample>** for binding the clusterRole to the managed service account.

1.7.3. Using a managed service account in the GitOpsCluster resource

The GitOpsCluster resource uses placement to import selected managed clusters into the Argo CD, including the creation of the Argo CD cluster secrets which contains information used to access the clusters. By default, the Argo CD cluster secret uses the application manager service account to access the managed clusters.

1. To update the GitOpsCluster resource to use the managed service account , add the **managedServiceAccountRef** property with the name of the managed service account.

2. To create a GitOpsCluster custom resource, save the following YAML as a Gitops.YAML:

```
---
apiVersion: apps.open-cluster-management.io/v1beta1
metadata:
  name: argo-acm-importer
  namespace: openshift-gitops
spec:
  managedServiceAccountRef: <managed-sa-sample>
  argoServer:
    cluster: notused
    argoNamespace: openshift-gitops
  placementRef:
    kind: Placement
    apiVersion: cluster.open-cluster-management.io/v1beta1
    name: all-openshift-clusters
    namespace: openshift-gitops
```

3. Save the YAML file in a file called, **gitops.yaml**.

4. Run **oc apply -f gitops.yaml**.

5. Go to the **openshift-gitops** namespace and verify that there is a new Argo CD cluster secret with the name **<managed cluster-managed-sa-sample-cluster-secret>**:

```
% oc get secrets -n openshift-gitops <managed cluster-managed-sa-sample-cluster-secret>
NAME                                                TYPE     DATA   AGE
<managed cluster-managed-sa-sample-cluster-secret>   Opaque   3      4m2s
```

1.7.4. Creating an Argo CD application

Deploy an Argo CD application from the Argo CD console by using the pushing model. The Argo CD application is deployed with the managed service account, **<managed-sa-sample>**.

1. Log into the Argo CD console.
2. Click **Create a new application**
3. Choose the cluster URL.
4. Go to your Argo CD application and verify that it has the given permissions, like roles and cluster roles, that you propagated to **<managed cluster>**.

1.7.5. Using policy to create managed service accounts and cluster permissions

> When the GitOpsCluster resource is updated with the `managedServiceAccountRef`, each managed cluster in the placement of this GitOpsCluster needs to have the service account. If you have several managed clusters, it becomes tedious for you to create the managed service account and cluster permission for each managed cluster. You can simply this process by using a policy to create the managed service account and cluster permission for all your managed clusters

When you apply the **managedServiceAccount** and **clusterPermission** resources to the hub cluster, the placement of this policy is bound to the local cluster. Replicate those resources to the managed cluster namespace for all of the managed clusters in the placement of the GitOpsCluster resource.

Using a policy to create the **managedServiceAccount** and **clusterPermission** resources include the following attributes:

- Updating the **managedServiceAccount** and **clusterPermission** object templates in the policy results in updates to all of the **managedServiceAccount** and **clusterPermission** resources in all of the managed clusters.
- Updating directly to the **managedServiceAccount** and **clusterPermission** resources becomes reverted back to the original state because it is enforced by the policy.
- If the placement decision for the GitOpsCluster placement changes, the policy manages the creation and deletion of the resources in the managed cluster namespaces.

1. To create a policy for a YAML to create a managed service account and cluster permission, create a YAML with the following content:

```
apiVersion: policy.open-cluster-management.io/v1
kind: Policy
metadata:
  name: policy-gitops
  namespace: openshift-gitops
  annotations:
```

```
    policy.open-cluster-management.io/standards: NIST-CSF
    policy.open-cluster-management.io/categories: PR.PT Protective Technology
    policy.open-cluster-management.io/controls: PR.PT-3 Least Functionality
spec:
  remediationAction: enforce
  disabled: false
  policy-templates:

    - objectDefinition:
        apiVersion: policy.open-cluster-management.io/v1
        kind: ConfigurationPolicy
        metadata:
          name: policy-gitops-sub
        spec:
          pruneObjectBehavior: None
          remediationAction: enforce
          severity: low
          object-templates-raw: |
            {{ range $placedec := (lookup "cluster.open-cluster-management.io/v1beta1"
"PlacementDecision" "openshift-gitops" "" "cluster.open-cluster-
management.io/placement=aws-app-placement").items }}
            {{ range $clustdec := $placedec.status.decisions }}
            - complianceType: musthave
              objectDefinition:
                apiVersion: authentication.open-cluster-management.io/v1alpha1
                kind: ManagedServiceAccount
                metadata:
                  name: <managed-sa-sample>
                  namespace: {{ $clustdec.clusterName }}
                spec:
                  rotation: {}
            - complianceType: musthave
              objectDefinition:
                apiVersion: rbac.open-cluster-management.io/v1alpha1
                kind: ClusterPermission
                metadata:
                  name: <clusterpermission-msa-subject-sample>
                  namespace: {{ $clustdec.clusterName }}
                spec:
                  roles:
                  - namespace: default
                    rules:
                    - apiGroups: ["apps"]
                      resources: ["deployments"]
                      verbs: ["get", "list", "create", "update", "delete"]
                    - apiGroups: [""]
                      resources: ["configmaps", "secrets", "pods", "podtemplates",
"persistentvolumeclaims", "persistentvolumes"]
                      verbs: ["get", "update", "list", "create", "delete"]
                    - apiGroups: ["storage.k8s.io"]
                      resources: ["*"]
                      verbs: ["list"]
                  - namespace: mortgage
                    rules:
                    - apiGroups: ["apps"]
                      resources: ["deployments"]
```

```
                verbs: ["get", "list", "create", "update", "delete"]
              - apiGroups: [""]
                resources: ["configmaps", "secrets", "pods", "services", "namespace"]
                verbs: ["get", "update", "list", "create", "delete"]
            clusterRole:
              rules:
              - apiGroups: ["*"]
                resources: ["*"]
                verbs: ["get", "list"]
            roleBindings:
            - namespace: default
              roleRef:
                kind: Role
              subject:
                apiGroup: authentication.open-cluster-management.io
                kind: ManagedServiceAccount
                name: <managed-sa-sample>
            - namespace: mortgage
              roleRef:
                kind: Role
              subject:
                apiGroup: authentication.open-cluster-management.io
                kind: ManagedServiceAccount
                name: <managed-sa-sample>
            clusterRoleBinding:
              subject:
                apiGroup: authentication.open-cluster-management.io
                kind: ManagedServiceAccount
                name: <managed-sa-sample>
        {{ end }}
        {{ end }}
---
apiVersion: policy.open-cluster-management.io/v1
kind: PlacementBinding
metadata:
  name: binding-policy-gitops
  namespace: openshift-gitops
placementRef:
  name: lc-app-placement
  kind: Placement
  apiGroup: cluster.open-cluster-management.io
subjects:
  - name: policy-gitops
    kind: Policy
    apiGroup: policy.open-cluster-management.io
---
apiVersion: cluster.open-cluster-management.io/v1beta1
kind: Placement
metadata:
  name: lc-app-placement
  namespace: openshift-gitops
spec:
  numberOfClusters: 1
  predicates:
  - requiredClusterSelector:
```

```
    labelSelector:
      matchLabels:
        name: local-cluster
```

1. Save the YAML file in a file called, **policy.yaml**.

2. Run **oc apply -f policy.yaml**.

3. In the object template of the policy, it iterates through the placement decision of the GitOpsCluster associated placement and applies the following **managedServiceAccount** and **clusterPermission** templates: