Red Hat Advanced Cluster Management for Kubernetes 2.6

Add-ons

Read more to learn how to use add-ons for your cluster.

Last Updated: 2023-10-31

Red Hat Advanced Cluster Management for Kubernetes 2.6 Add-ons

Read more to learn how to use add-ons for your cluster.

Legal Notice

Abstract

Read more to learn how to use add-ons for your cluster.

Table of Contents

CHAPTER 1. ADD-ONS OVERVIEW

Red Hat Advanced Cluster Management for Kubernetes add-ons can improve some areas of performance and add function to enhace your applications. The following sections provide a summary of the add-ons that are available for Red Hat Advanced Cluster Management:

- Submariner multicluster networking and service discovery
- VolSync persistent volume replicating service
- Enabling klusterlet add-ons on clusters from the multicluster engine for Kubernetes operator

1.1. SUBMARINER MULTICLUSTER NETWORKING AND SERVICE DISCOVERY

Submariner is an open source tool that can be used with Red Hat Advanced Cluster Management for Kubernetes to provide direct networking and service discovery between two or more managed clusters in your environment, either on-premises or in the cloud. Submariner is compatible with Multi-Cluster Services API (Kubernetes Enhancements Proposal #1645). For more information about Submariner, see the Submariner site.

Red Hat Advanced Cluster Management for Kubernetes provides Submariner as an add-on for your hub cluster. You can find more information about Submariner in the Submariner open source project documentation.

See the Red Hat Advanced Cluster Management Support Matrix for more details about which infrastructure providers are supported by automated console deployments, and which infrastructure providers require manual deployment.

- Prerequisites
- subctl command utility
- Globalnet
- Deploying Submariner
  - Deploying Submariner with the console
  - Deploying Submariner manually
    - Preparing selected hosts for Submariner
    - Deploy Submariner with the ManagedClusterAddOn API
    - Customizing Submariner deployments
- Managing service discovery for Submariner
- Uninstalling Submariner

1.1.1. Prerequisites

Ensure that you have the following prerequisites before using Submariner:

- A credential to access the hub cluster with **cluster-admin** permissions.

- IP connectivity must be configured between the gateway nodes. When connecting two clusters, at least one of the clusters must be accessible to the gateway node using its public or private IP address designated to the gateway node. See Submariner NAT Traversal for more information.
- If you are using OVN Kubernetes, your clusters must be at Red Hat OpenShift Container Platform version 4.11 or later.
- Firewall configuration across all nodes in each of the managed clusters must allow 4800/UDP in both directions.
- Firewall configuration on the gateway nodes must allow ingress 8080/TCP so the other nodes in the cluster can access it.
- Firewall configuration open for 4500/UDP and any other ports that are used for IPsec traffic on the gateway nodes.
- If the gateway nodes are directly reachable over their private IPs without any NAT in between, make sure that the firewall configuration allows the ESP protocol on the gateway nodes.
  Note: This is configured automatically when your clusters are deployed in an AWS or GCP environment, but must be configured manually for clusters on other environments and for the firewalls that protect private clouds.
- The **managedcluster** name must follow the DNS label standard as defined in RFC 1123. This means the name must meet the following criteria:
  - Contain at most 63 characters
  - Contain only lowercase alphanumeric characters or '-'
  - Start with an alphanumeric character
  - End with an alphanumeric character

1.1.2. Submariner ports table

View the following table to see which Submariner ports you need to enable:

Name	Default value	Customizable	Optional or required
IPsec NATT	4500/UDP	Yes	Required
VXLAN	4800/UDP	No	Required
Submariner metrics port	8080/TCP	No	Required
Globalnet metrics port	8081/TCP	No	Required when Globalnet is enabled

See the Submariner upstream prerequisites documentation for more detailed information about the prerequisites.

1.1.3. subctl command utility

Submariner contains the **subctl** utility that provides additional commands that simplify running tasks on your Submariner environment.

1.1.3.1. Installing the subctl command utility

The **subctl** utility is shipped in a container image. Complete the following steps to install the **subctl** utility locally:

1. Log in to the registry by running the following command and entering your credentials when prompted:

```
oc registry login --registry registry.redhat.io
```

2. Download the **subctl** container and extract a compressed version of the **subctl** binary to **/tmp** by entering the following command:

```
oc image extract registry.redhat.io/rhacm2/subctl-rhel8:v0.13 --path="/dist/subctl-v0.13*-linux-amd64.tar.xz":/tmp/ --confirm
```

3. Decompress the **subctl** utility by entering the following command:

```
tar -C /tmp/ -xf /tmp/subctl-v0.13*-linux-amd64.tar.xz
```

4. Install the **subctl** utility by entering the following command:

```
install -m744 /tmp/subctl-v0.13*/subctl-v0.13*-linux-amd64 /$HOME/.local/bin/subctl
```

1.1.3.2. Using the subctl commands

After adding the utility to your path, view the following table for a brief description of the available commands:

export service	Creates a **ServiceExport** resource for the specified service, which enables other clusters in the Submariner deployment to discover the corresponding service.
unexport service	Removes the **ServiceExport** resource for the specified service, which prevents other clusters in the Submariner deployment from discovering the corresponding service.
show	Provides information about Submariner resources.
verify	Verifies connectivity, service discovery, and other Submariner features when Submariner is configured across a pair of clusters.
benchmark	Benchmarks throughput and latency across a pair of clusters that are enabled with Submariner or within a single cluster.

diagnose	Runs checks to identify issues that prevent the Submariner deployment from working correctly.
gather	Collects information from the clusters to help troubleshoot a Submariner deployment.
version	Displays the version details of the **subctl** binary tool.

For more information about the **subctl** utility and its commands, see **subctl** in the Submariner documentation.

1.1.4. Globalnet

Globalnet is a feature included with the Submariner add-on which supports connectivity between clusters with overlapping CIDRs. Globalnet is a cluster set wide configuration, and can be selected when the first managed cluster is added to the cluster set. When Globalnet is enabled, each managed cluster is allocated a global CIDR from the virtual Global Private Network. The global CIDR is used for supporting inter-cluster communication.

If there is a chance that your clusters running Submariner might have overlapping CIDRs, consider enabling Globalnet. When using the console, the **ClusterAdmin** can enable Globalnet for a cluster set by selecting the option **Enable Globalnet** when enabling the Submariner add-on for clusters in the cluster set. After you enable Globalnet, you cannot disable it without removing Submariner.

When using the Red Hat Advanced Cluster Management APIs, the **ClusterAdmin** can enable Globalnet by creating a **submariner-broker** object in the **<ManagedClusterSet>-broker** namespace.

The **ClusterAdmin** role has the required permissions to create this object in the broker namespace. The **ManagedClusterSetAdmin** role, which is sometimes created to act as a proxy administrator for the cluster set, does not have the required permissions. To provide the required permissions, the **ClusterAdmin** must associate the role permissions for the **access-to-brokers-submariner-crd** to the **ManagedClusterSetAdmin** user.

Complete the following steps to create the **submariner-broker** object:

1. Retrieve the **<broker-namespace>** by running the following command:

```
oc get ManagedClusterSet <cluster-set-name> -o jsonpath="
{.metadata.annotations['cluster\.open-cluster-management\.io/submariner-broker-ns']}"
```

2. Create a **submariner-broker** object that specifies the Globalnet configuration by creating a YAML file named **submariner-broker**. Add content that resembles the following lines to the YAML file:

```
apiVersion: submariner.io/v1alpha1
kind: Broker
metadata:
  name: submariner-broker
  namespace: <broker-namespace>
spec:
  globalnetEnabled: <true-or-false>
```

   Replace **broker-namespace** with the name of your broker namespace.

Replace **true-or-false** with **true** to enable Globalnet.

Note: The **metadata name** parameter must be **submariner-broker**.

3. Apply the file to your YAML file by entering the following command:

```
oc apply -f submariner-broker.yaml
```

For more information about Globalnet, see Globalnet controller in the Submariner documentation.

1.1.5. Deploying Submariner

You can deploy Submariner to network clusters on the following providers:

Automatic deployment process:

- Amazon Web Services
- Google Cloud Platform
- Red Hat OpenStack Platform
- Microsoft Azure

Manual deployment process:

- IBM Cloud
- VMware vSphere
- Bare metal

1.1.5.1. Deploying Submariner with the console

You can deploy Submariner on Red Hat OpenShift Container Platform managed clusters that are deployed on Amazon Web Services, Google Cloud Platform, and VMware vSphere by using the Red Hat Advanced Cluster Management for Kubernetes console. To deploy Submariner on other providers, follow the instructions in Deploying Submariner manually. Complete the following steps to deploy Submariner with the Red Hat Advanced Cluster Management for Kubernetes console:

Required access: Cluster administrator

1. From the console navigation menu, select **Infrastructure** > **Clusters**.
2. On the *Clusters* page, select the *Cluster sets* tab. The clusters that you want enable with Submariner must be in the same cluster set.
3. If the clusters on which you want to deploy Submariner are already in the same cluster set, skip to step 5 to deploy Submariner.
4. If the clusters on which you want to deploy Submariner are not in the same cluster set, create a cluster set for them by completing the following steps:
   a. Select **Create cluster set**
   b. Name the cluster set, and select **Create**.

c. Select **Manage resource assignments** to assign clusters to the cluster set.

d. Select the managed clusters that you want to connect with Submariner to add them to the cluster set.

e. Select **Review** to view and confirm the clusters that you selected.

f. Select **Save** to save the cluster set, and view the resulting cluster set page.

5. On the cluster set page, select the *Submariner add-ons* tab.

6. Select **Install Submariner add-ons**.

7. Select the clusters on which you want to deploy Submariner.

8. Enter the following information in the *Install Submariner add-ons* editor:

   - **AWS Access Key ID** - This field is only visible when you import an AWS cluster.
   - **AWS Secret Access Key** - This field is only visible when you import an AWS cluster.
   - **Google Cloud Platform service account JSON key** - This field is only visible when you import a Google Cloud Platform cluster.
   - **Instance type** - The Amazon Web Services EC2 instance type of the gateway node that is created on the managed cluster. The default value is **c5d.large**. This field is only visible when your managed cluster environment is AWS.
   - **IPsec NAT-T port** - The default value for the IPsec NAT traversal port is port **4500**. If your managed cluster environment is VMware vSphere, ensure that this port is opened on your firewalls.
   - **Gateway count** - The number of worker nodes that are used to deploy the Submariner gateway component on your managed cluster. The default value is **1**. If the value is greater than 1, the Submariner gateway High Availability (HA) is automatically enabled.
   - **Cable driver** - The Submariner gateway cable engine component that maintains the cross-cluster tunnels. The default value is **Libreswan IPsec**.

9. Select **Next** at the end of the editor to move to the editor for the next cluster, and complete the editor for each of the remaining clusters that you selected.

10. Verify your configuration for each managed cluster.

11. Click **Install** to deploy Submariner on the selected managed clusters.
    It might take several minutes for the installation and configuration to complete. You can check the Submariner status in the list on the *Submariner add-ons* tab:

    - **Connection status** indicates how many Submariner connections are established on the managed cluster.
    - **Agent status** indicates whether Submariner is successfully deployed on the managed cluster. The console might report a status of **Degraded** until it is installed and configured.
    - **Gateway nodes labeled** indicates how many worker nodes are labeled with the Submariner gateway label: **submariner.io/gateway=true** on the managed cluster.

Submariner is now deployed on the clusters.

1.1.5.2. Deploying Submariner manually

Before you deploy Submariner with Red Hat Advanced Cluster Management for Kubernetes, you must prepare the clusters on the hosting environment for the connection. Currently, you can use the **SubmarinerConfig** API to automatically prepare the clusters on Amazon Web Services, Google Cloud Platform and VMware vSphere. For other platforms, you need to prepare them manually, see Preparing selected hosts to deploy Submariner for the steps.

1.1.5.2.1. Preparing selected hosts to deploy Submariner

Before you deploy Submariner with Red Hat Advanced Cluster Management for Kubernetes, you must manually prepare the clusters on the hosting environment for the connection. The requirements are different for different hosting environments, so follow the instructions for your hosting environment.

1.1.5.2.1.1. Preparing VMware vSphere for Submariner

Submariner uses IPsec to establish the secure tunnels between the clusters on the gateway nodes. You can use the default port or specify a custom port. When you run this procedure without specifying an IPsec NATT port, the default port is automatically used for the communication. The default port is 4500/UDP.

Submariner uses virtual extensible LAN (VXLAN) to encapsulate traffic when it moves from the worker and master nodes to the gateway nodes. The VXLAN port cannot be customized, and is always port 4800/UDP.

Submariner uses 8080/TCP to send its metrics information among nodes in the cluster, this port cannot be customized.

The following ports must be opened by your VMWare vSphere administrator before you can enable Submariner:

Table 1.1. VMware vSphere and Submariner ports

Name	Default value	Customizable
IPsec NATT	4500/UDP	Yes
VXLAN	4800/UDP	No
Submariner metrics	8080/TCP	No

To prepare VMware vSphere clusters for deploying Submariner, complete the following steps:

1. Ensure that the IPsec NATT, VXLAN, and metrics ports are open.
2. Customize and apply YAML content that is similar to the following example:

```
apiVersion: submarineraddon.open-cluster-management.io/v1alpha1
kind: SubmarinerConfig
metadata:
   name: submariner
   namespace: <managed-cluster-namespace>
spec:{}
```

Replace **managed-cluster-namespace** with the namespace of your managed cluster.

Note: The name of the **SubmarinerConfig** must be **submariner**, as shown in the example.

This configuration uses the default network address translation - traversal (NATT) port (4500/UDP) for your Submariner and one worker node is labeled as the Submariner gateway on your vSphere cluster.

Submariner uses IP security (IPsec) to establish the secure tunnels between the clusters on the gateway nodes. You can either use the default IPsec NATT port, or you can specify a different port that you configured. When you run this procedure without specifying an IPsec NATT port of 4500/UDP is automatically used for the communication.

1.1.5.2.1.2. Preparing bare metal for Submariner

To prepare bare metal clusters for deploying Submariner, complete the following steps:

1. Ensure that the IPsec NATT, VXLAN, and metrics ports are open.
2. Customize and apply YAML content that is similar to the following example:

```
apiVersion: submarineraddon.open-cluster-management.io/v1alpha1
kind: SubmarinerConfig
metadata:
   name: submariner
   namespace: <managed-cluster-namespace>
spec:{}
```

Replace **managed-cluster-namespace** with the namespace of your managed cluster.

Note: The name of the **SubmarinerConfig** must be **submariner**, as shown in the example.

This configuration uses the default network address translation - traversal (NATT) port (4500/UDP) for your Submariner and one worker node is labeled as the Submariner gateway on your bare metal cluster.

Submariner uses IP security (IPsec) to establish the secure tunnels between the clusters on the gateway nodes. You can either use the default IPsec NATT port, or you can specify a different port that you configured. When you run this procedure without specifying an IPsec NATT port of 4500/UDP is automatically used for the communication.

See Customizing Submariner deployments for information about the customization options.

1.1.5.2.2. Deploy Submariner with the ManagedClusterAddOn API

To deploy Submariner by using the **ManagedClusterAddOn** API, you must first prepare the clusters on the hosting environment. See Preparing selected hosts to deploy Submariner for more details.

After preparing the clusters, complete the following steps:

1. Create a **ManagedClusterSet** resource on the hub cluster by using the instructions provided in the *Creating and managing ManagedClusterSets* topic of the Creating and managing ManagedClusterSets documentation. Your entry for the **ManagedClusterSet** should resemble the following content:

```
apiVersion: cluster.open-cluster-management.io/v1beta1
```

```
kind: ManagedClusterSet
metadata:
  name: <managed-cluster-set-name>
```

Replace **managed-cluster-set-name** with a name for the **ManagedClusterSet** that you are creating.

Note: The maximum length of the name of the Kubernetes namespace is 63 characters, so the maximum length of the **<managed-cluster-set-name>** is 56 characters. If the length of **<managed-cluster-set-name>** exceeds 56, the **<managed-cluster-set-name>** is truncated from the head.

After the **ManagedClusterSet** is created, the **submariner-addon** creates a namespace called **<managed-cluster-set-name>-broker** and deploys the Submariner broker to it.

2. Create the **Broker** configuration on the hub cluster in the **<managed-cluster-set-name>-broker** namespace by customizing and applying YAML content that is similar to the following example:

```
apiVersion: submariner.io/v1alpha1
kind: Broker
metadata:
    name: submariner-broker
    namespace: <managed-cluster-set-name>-broker
spec:
    globalnetEnabled: false
```

Replace **managed-cluster-set-name** with the name of the managed cluster.

Set the the value of **globalnetEnabled** to **true** if you want to enable Submariner Globalnet in the **ManagedClusterSet**.

3. Add one managed cluster to the **ManagedClusterSet** by entering the following command:

```
oc label managedclusters <managed-cluster-name> "cluster.open-cluster-
management.io/clusterset=<managed-cluster-set-name>" --overwrite
```

Replace **<managed-cluster-name>** with the name of the managed cluster that you want to add to the **ManagedClusterSet**.

Replace **<managed-cluster-set-name>** with the name of the **ManagedClusterSet** to which you want to add the managed cluster.

4. Customize and apply YAML content that is similar to the following example:

```
apiVersion: submarineraddon.open-cluster-management.io/v1alpha1
kind: SubmarinerConfig
metadata:
   name: submariner
   namespace: <managed-cluster-namespace>
spec:{}
```

Replace **managed-cluster-namespace** with the namespace of your managed cluster.

Note: The name of the **SubmarinerConfig** must be **submariner**, as shown in the example.

5. Deploy Submariner on the managed cluster by customizing and applying YAML content that is similar to the following example:

```
apiVersion: addon.open-cluster-management.io/v1alpha1
kind: ManagedClusterAddOn
metadata:
    name: submariner
    namespace: <managed-cluster-name>
spec:
    installNamespace: submariner-operator
```

Replace **managed-cluster-name** with the name of the managed cluster that you want to use with Submariner.

The **installNamespace** field in the spec of the **ManagedClusterAddOn** is the namespace on the managed cluster where it installs Submariner. Currently, Submariner must be installed in the **submariner-operator** namespace.

After the **ManagedClusterAddOn** is created, the **submariner-addon** deploys Submariner to the **submariner-operator** namespace on the managed cluster. You can view the deployment status of Submariner from the status of this **ManagedClusterAddOn**.

Note: The name of **ManagedClusterAddOn** must be **submariner**.

6. Repeat steps three, four, and five for all of the managed clusters that you want to enable Submariner on.

7. After Submariner is deployed on the managed cluster, you can verify the Submariner deployment status by checking the status of submariner **ManagedClusterAddOn** by entering the following command:

```
oc -n <managed-cluster-name> get managedclusteraddons submariner -oyaml
```

Replace **managed-cluster-name** with the name of the managed cluster.

In the status of the Submariner **ManagedClusterAddOn**, three conditions indicate the deployment status of Submariner:

- **SubmarinerGatewayNodesLabeled** condition indicates whether there are labeled Submariner gateway nodes on the managed cluster.
- **SubmarinerAgentDegraded** condition indicates whether the Submariner is successfully deployed on the managed cluster.
- **SubmarinerConnectionDegraded** condition indicates how many connections are established on the managed cluster with Submariner.

1.1.5.2.3. Customizing Submariner deployments

You can customize some of the settings of your Submariner deployments, including your Network Address Translation-Traversal (NATT) port, number of gateway nodes, and instance type of your gateway nodes. These customizations are consistent across all of the providers.

1.1.5.2.3.1. NATT port

If you want to customize your NATT port, customize and apply the following YAML content for your provider environment:

```
apiVersion: submarineraddon.open-cluster-management.io/v1alpha1
kind: SubmarinerConfig
metadata:
  name: submariner
  namespace: <managed-cluster-namespace>
spec:
  credentialsSecret:
    name: <managed-cluster-name>-<provider>-creds
  IPSecNATTPort: <NATTPort>
```

- Replace **managed-cluster-namespace** with the namespace of your managed cluster.
- Replace **managed-cluster-name** with the name of your managed cluster
  - AWS: Replace **provider** with **aws**. The value of **<managed-cluster-name>-aws-creds** is your AWS credential secret name, which you can find in the cluster namespace of your hub cluster.
  - GCP: Replace **provider** with **gcp**. The value of **<managed-cluster-name>-gcp-creds** is your Google Cloud Platform credential secret name, which you can find in the cluster namespace of your hub cluster.
  - OpenStack: Replace **provider** with **osp**. The value of **<managed-cluster-name>-osp-creds** is your Red Hat OpenStack Platform credential secret name, which you can find in the cluster namespace of your hub cluster.
  - Azure: Replace **provider** with **azure**. The value of **<managed-cluster-name>-azure-creds** is your Microsoft Azure credential secret name, which you can find in the cluster namespace of your hub cluster.
- Replace **managed-cluster-namespace** with the namespace of your managed cluster.
- Replace **managed-cluster-name** with the name of your managed cluster. The value of **managed-cluster-name-gcp-creds** is your Google Cloud Platform credential secret name, which you can find in the cluster namespace of your hub cluster.
- Replace **NATTPort** with the NATT port that you want to use.

Note: The name of the **SubmarinerConfig** must be **submariner**, as shown in the example.

To customize your NATT port in the VMware vSphere environment, customize and apply the following YAML content:

```
apiVersion: submarineraddon.open-cluster-management.io/v1alpha1
kind: SubmarinerConfig
metadata:
  name: submariner
  namespace: <managed-cluster-namespace>
spec:
  IPSecNATTPort: <NATTPort>
```

- Replace **managed-cluster-namespace** with the namespace of your managed cluster.
- Replace **NATTPort** with the NATT port that you want to use.

Note: The name of the **SubmarinerConfig** must be **submariner**, as shown in the example.

1.1.5.2.3.2. Number of gateway nodes

If you want to customize the number of your gateway nodes, customize and apply YAML content that is similar to the following example:

```
apiVersion: submarineraddon.open-cluster-management.io/v1alpha1
kind: SubmarinerConfig
metadata:
  name: submariner
  namespace: <managed-cluster-namespace>
spec:
  credentialsSecret:
    name: <managed-cluster-name>-<provider>-creds
 gatewayConfig:
    gateways: <gateways>
```

- Replace **managed-cluster-namespace** with the namespace of your managed cluster.
- Replace **managed-cluster-name** with the name of your managed cluster.
  - AWS: Replace **provider** with **aws**. The value of **<managed-cluster-name>-aws-creds** is your AWS credential secret name, which you can find in the cluster namespace of your hub cluster.
  - GCP: Replace **provider** with **gcp**. The value of **<managed-cluster-name>-gcp-creds** is your Google Cloud Platform credential secret name, which you can find in the cluster namespace of your hub cluster.
  - OpenStack: Replace **provider** with **osp**. The value of **<managed-cluster-name>-osp-creds** is your Red Hat OpenStack Platform credential secret name, which you can find in the cluster namespace of your hub cluster.
  - Azure: Replace **provider** with **azure**. The value of **<managed-cluster-name>-azure-creds** is your Microsoft Azure credential secret name, which you can find in the cluster namespace of your hub cluster.
- Replace **gateways** with the number of gateways that you want to use. If the value is greater than 1, the Submariner gateway automatically enables high availability.

Note: The name of the **SubmarinerConfig** must be **submariner**, as shown in the example.

If you want to customize the number of your gateway nodes in the VMware vSphere environment, customize and apply YAML content that is similar to the following example:

```
apiVersion: submarineraddon.open-cluster-management.io/v1alpha1
kind: SubmarinerConfig
metadata:
  name: submariner
  namespace: <managed-cluster-namespace>
spec:
 gatewayConfig:
    gateways: <gateways>
```

- Replace **managed-cluster-namespace** with the namespace of your managed cluster.

- Replace **gateways** with the number of gateways that you want to use. If the value is greater than 1, the Submariner gateway automatically enables high availability.

1.1.5.2.3.3. Instance types of gateway nodes

If you want to customize the instance type of your gateway node, customize and apply YAML content that is similar to the following example:

```
apiVersion: submarineraddon.open-cluster-management.io/v1alpha1
kind: SubmarinerConfig
metadata:
  name: submariner
  namespace: <managed-cluster-namespace>
spec:
  credentialsSecret:
    name: <managed-cluster-name>-<provider>-creds
 gatewayConfig:
    instanceType: <instance-type>
```

- Replace **managed-cluster-namespace** with the namespace of your managed cluster.
- Replace **managed-cluster-name** with the name of your managed cluster.
  - AWS: Replace **provider** with **aws**. The value of **<managed-cluster-name>-aws-creds** is your AWS credential secret name, which you can find in the cluster namespace of your hub cluster.
  - GCP: Replace **provider** with **gcp**. The value of **<managed-cluster-name>-gcp-creds** is your Google Cloud Platform credential secret name, which you can find in the cluster namespace of your hub cluster.
  - OpenStack: Replace **provider** with **osp**. The value of **<managed-cluster-name>-osp-creds** is your Red Hat OpenStack Platform credential secret name, which you can find in the cluster namespace of your hub cluster.
  - Azure: Replace **provider** with **azure**. The value of **<managed-cluster-name>-azure-creds** is your Microsoft Azure credential secret name, which you can find in the cluster namespace of your hub cluster.
- Replace **instance-type** with the AWS instance type that you want to use.

Note: The name of the **SubmarinerConfig** must be **submariner**, as shown in the example.

1.1.5.2.3.4. Cable driver

The Submariner Gateway Engine component creates secure tunnels to other clusters. The cable driver component maintains the tunnels by using a pluggable architecture in the Gateway Engine component. You can use the Libreswan or VXLAN implementations for the **cableDriver** configuration of the cable engine component. See the following example:

```
apiVersion: submarineraddon.open-cluster-management.io/v1alpha1
kind: SubmarinerConfig
metadata:
  name: submariner
  namespace: <managed-cluster-namespace>
spec:
```

```
cableDriver: vxlan
credentialsSecret:
  name: <managed-cluster-name>-<provider>-creds
```

Best practice: Do not use the VXLAN cable driver on public networks. The VXLAN cable driver is unencrypted. Only use VXLAN to avoid unnecessary double encryption on private networks. For example, some on-premise environments might handle the tunnel's encryption with a dedicated line-level hardware device.

1.1.5.3. Managing service discovery for Submariner

After Submariner is deployed into the same environment as your managed clusters, the routes are configured for secure IP routing between the pod and services across the clusters in the managed cluster set.

1.1.5.3.1. Enabling service discovery for Submariner

To make a service from a cluster visible and discoverable to other clusters in the managed cluster set, you must create a **ServiceExport** object. After a service is exported with a **ServiceExport** object, you can access the service by the following format: **<service>.<namespace>.svc.clusterset.local**. If multiple clusters export a service with the same name, and from the same namespace, they are recognized by other clusters as a single logical service.

This example uses the **nginx** service in the **default** namespace, but you can discover any Kubernetes **ClusterIP** service or headless service:

1. Apply an instance of the **nginx** service on a managed cluster that is in the **ManagedClusterSet** by entering the following commands:

   ```
   oc -n default create deployment nginx --image=nginxinc/nginx-unprivileged:stable-alpine
   oc -n default expose deployment nginx --port=8080
   ```

2. Export the service by creating a **ServiceExport** entry by entering a command with the **subctl** tool that is similar to the following command:

   ```
   subctl export service --namespace <service-namespace> <service-name>
   ```

   Replace **service-namespace** with the name of the namespace where the service is located. In this example, it is **default**.

   Replace **service-name** with the name of the service that you are exporting. In this example, it is **nginx**.

   See **export** in the Submariner documentation for more information about other available flags.

3. Run the following command from a different managed cluster to confirm that it can access the **nginx** service:

   ```
   oc -n default run --generator=run-pod/v1 tmp-shell --rm -i --tty --image
   quay.io/submariner/nettest -- /bin/bash curl nginx.default.svc.clusterset.local:8080
   ```

The **nginx** service discovery is now configured for Submariner.

1.1.5.3.2. Disabling service discovery for Submariner

To disable a service from being exported to other clusters, enter a command similar to the following example for **nginx**:

```
subctl unexport service --namespace <service-namespace> <service-name>
```

Replace **service-namespace** with the name of the namespace where the service is located.

Replace **service-name** with the name of the service that you are exporting.

See **unexport** in the Submariner documentation for more information about other available flags.

The service is no longer available for discovery by clusters.

1.1.5.4. Uninstalling Submariner

You can uninstall the Submariner components from your clusters using the Red Hat Advanced Cluster Management for Kubernetes console or the command-line. For Submariner versions earlier than 0.12, additional steps are needed to completely remove all data plane components. The Submariner uninstall is idempotent, so you can repeat steps without any issues.

1.1.5.4.1. Console method

To uninstall Submariner from a cluster by using the console, complete the following steps:

1. From the console navigation, select **Infrastructure** > **Clusters**, and select the *Cluster sets* tab.
2. Select the cluster set that contains the clusters from which you want to remove the Submariner components.
3. Select the **Submariner Add-ons** tab to view the clusters in the cluster set that have Submariner deployed.
4. In the *Actions* menu for the cluster that you want to uninstall Submariner, select **Uninstall Add-on**.
5. In the *Actions* menu for the cluster that you want to uninstall Submariner, select **Delete cluster sets**.
6. Repeat those steps for other clusters from which you are removing Submariner.
   Tip: You can remove the Submariner add-on from multiple clusters in the same cluster set by selecting multiple clusters and clicking **Actions**. Select **Uninstall Submariner add-ons**.

If the version of Submariner that you are removing is earlier than version 0.12, continue with Manual removal steps for early versions of Submariner. If the Submariner version is 0.12, or later, Submariner is removed.

Important: Verify that all of the cloud resources are removed from the cloud provider to avoid additional charges by your cloud provider. See Verifying Submariner resource removal for more information.

1.1.5.4.2. Command-line method

To uninstall Submariner by using the command line, complete the following steps:

1. Remove the Submariner deployment for the cluster by running the following command:

   ```
   oc -n <managed-cluster-namespace> delete managedclusteraddon submariner
   ```

Replace **managed-cluster-namespace** with the namespace of your managed cluster.

2. Remove the cloud resources of the cluster by running the following command:

```
oc -n <managed-cluster-namespace> delete submarinerconfig submariner
```

Replace **managed-cluster-namespace** with the namespace of your managed cluster.

3. Delete the cluster set to remove the broker details by running the following command:

```
oc delete managedclusterset <managedclusterset>
```

Replace **managedclusterset** with the name of your managed cluster set.

If the version of Submariner that you are removing is earlier than version 0.12, continue with Manual removal steps for early versions of Submariner. If the Submariner version is 0.12, or later, Submariner is removed.

Important: Verify that all of the cloud resources are removed from the cloud provider to avoid additional charges by your cloud provider. See Verifying Submariner resource removal for more information.

1.1.5.4.3. Manual removal steps for early versions of Submariner

When uninstalling versions of Submariner that are earlier than version 0.12, complete steps 5-8 in the Manual Uninstall section in the Submariner documentation.

After completing those steps, your Submariner components are removed from the cluster.

Important: Verify that all of the cloud resources are removed from the cloud provider to avoid additional charges by your cloud provider. See Verifying Submariner resource removal for more information.

1.1.5.4.4. Verifying Submariner resource removal

After uninstalling Submariner, verify that all of the Submariner resources are removed from your clusters. If they remain on your clusters, some resources continue to accrue charges from infrastructure providers. Ensure that you have no additional Submariner resourceson your cluster by completing the following steps:

1. Run the following command to list any Submariner resources that remain on the cluster:

```
oc get cluster <CLUSTER_NAME> grep submariner
```

Replace **CLUSTER_NAME** with the name of your cluster.

2. Remove any resources on the list by entering the following command:

```
oc delete resource <RESOURCE_NAME> cluster <CLUSTER_NAME>
```

Replace **RESOURCE_NAME** with the name of the Submariner resource that you want to remove.

3. Repeat steps 1-2 for each of the clusters until your search does not identify any resources.

The Submariner resources are removed from your cluster.

1.2. VOLSYNC PERSISTENT VOLUME REPLICATION SERVICE

VolSync is a Kubernetes operator that enables asynchronous replication of persistent volumes within a cluster, or across clusters with storage types that are not otherwise compatible for replication. It uses the Container Storage Interface (CSI) to overcome the compatibility limitation. After deploying the VolSync operator in your environment, you can leverage it to create and maintain copies of your persistent data. VolSync can only replicate persistent volume claims on Red Hat OpenShift Container Platform clusters that are at version 4.8 or later.

- Replicating persistent volumes with VolSync
  - Installing VolSync on the managed clusters
  - Configuring an Rsync replication
  - Configuring a restic backup
  - Configuring an Rclone replication
- Converting a replicated image to a usable persistent volume claim
- Scheduling your synchronization

1.2.1. Replicating persistent volumes with VolSync

You can use three supported methods to replicate persistent volumes with VolSync, which depend on the number of synchronization locations that you have: Rsync, restic, or Rclone.

1.2.1.1. Prerequisites

Before installing VolSync on your clusters, you must have the following requirements:

- A configured Red Hat OpenShift Container Platform environment running a Red Hat Advanced Cluster Management version 2.4, or later, hub cluster
- At least two configured clusters that are managed by the same Red Hat Advanced Cluster Management hub cluster
- Network connectivity between the clusters that you are configuring with VolSync. If the clusters are not on the same network, you can configure the Submariner multicluster networking and service discovery and use the **ClusterIP** value for **ServiceType** to network the clusters, or use a load balancer with the **LoadBalancer** value for **ServiceType**.
- The storage driver that you use for your source persistent volume must be CSI-compatible and able to support snapshots.

1.2.1.2. Installing VolSync on the managed clusters

To enable VolSync to replicate the persistent volume claim on one cluster to the persistent volume claim of another cluster, you must install VolSync on both the source and the target managed clusters.

VolSync does not create its own namespace, so it is in the same namespace as other OpenShift Container Platform all-namespace operators. Any changes that you make to the operator settings for VolSync also affects the other operators in the same namespace, such as if you change to manual approval for channel updates.

You can use either of two methods to install VolSync on two clusters in your environment. You can either add a label to each of the managed clusters in the hub cluster, or you can manually create and apply a **ManagedClusterAddOn**, as they are described in the following sections:

1.2.1.2.1. Installing VolSync using labels

To install VolSync on the managed cluster by adding a label.

- Complete the following steps from the Red Hat Advanced Cluster Management console:
  1. Select one of the managed clusters from the **Clusters** page in the hub cluster console to view its details.
  2. In the **Labels** field, add the following label:

     ```
     addons.open-cluster-management.io/volsync=true
     ```

     The VolSync service pod is installed on the managed cluster.
  3. Add the same label the other managed cluster.
  4. Run the following command on each managed cluster to confirm that the VolSync operator is installed:

     ```
     oc get csv -n openshift-operators
     ```

     There is an operator listed for VolSync when it is installed.
- Complete the following steps from the command-line interface:
  1. Start a command-line session on the hub cluster.
  2. Enter the following command to add the label to the first cluster:

     ```
     oc label managedcluster <managed-cluster-1> "addons.open-cluster-
     management.io/volsync"="true"
     ```

     Replace **managed-cluster-1** with the name of one of your managed clusters.
  3. Enter the following command to add the label to the second cluster:

     ```
     oc label managedcluster <managed-cluster-2> "addons.open-cluster-
     management.io/volsync"="true"
     ```

     Replace **managed-cluster-2** with the name of your other managed cluster.

     A **ManagedClusterAddOn** resource should be created automatically on your hub cluster in the namespace of each corresponding managed cluster.

1.2.1.2.2. Installing VolSync using a ManagedClusterAddOn

To install VolSync on the managed cluster by adding a **ManagedClusterAddOn** manually, complete the following steps:

1. On the hub cluster, create a YAML file called **volsync-mcao.yaml** that contains content that is similar to the following example:

```
apiVersion: addon.open-cluster-management.io/v1alpha1
kind: ManagedClusterAddOn
metadata:
  name: volsync
  namespace: <managed-cluster-1-namespace>
spec: {}
```

Replace **managed-cluster-1-namespace** with the namespace of one of your managed clusters. This namespace is the same as the name of the managed cluster.

Note: The name must be **volsync**.

2. Apply the file to your configuration by entering a command similar to the following example:

```
oc apply -f volsync-mcao.yaml
```

3. Repeat the procedure for the other managed cluster.
A **ManagedClusterAddOn** resource should be created automatically on your hub cluster in the namespace of each corresponding managed cluster.

1.2.1.3. Configuring an Rsync replication

You can create a 1:1 asynchronous replication of persistent volumes by using an Rsync replication. You can use Rsync-based replication for disaster recovery or sending data to a remote site.

The following example shows how to configure by using the Rsync method. For additional information about Rsync, see Usage in the VolSync documentation.

1.2.1.3.1. Configuring Rsync replication across managed clusters

For Rsync-based replication, configure custom resources on the source and destination clusters. The custom resources use the **address** value to connect the source to the destination, and the **sshKeys** to ensure that the transferred data is secure.

Note: You must copy the values for **address** and **sshKeys** from the destination to the source, so configure the destination before you configure the source.

This example provides the steps to configure an Rsync replication from a persistent volume claim on the **source** cluster in the **source-ns** namespace to a persistent volume claim on a **destination** cluster in the **destination-ns** namespace. You can replace those values with other values, if necessary.

1. Configure your destination cluster.

   a. Run the following command on the destination cluster to create the namespace:

   ```
   oc create ns <destination-ns>
   ```

   Replace **destination-ns** with a name for the namespace that will contain your destination persistent volume claim.

   b. Copy the following YAML content to create a new file called **replication_destination.yaml**:

   ```
   apiVersion: volsync.backube/v1alpha1
   kind: ReplicationDestination
   metadata:
   ```

```
    name: <destination>
    namespace: <destination-ns>
  spec:
    rsync:
      serviceType: LoadBalancer
      copyMethod: Snapshot
      capacity: 2Gi
      accessModes: [ReadWriteOnce]
      storageClassName: gp2-csi
      volumeSnapshotClassName: csi-aws-vsc
```

Note: The **capacity** value should match the capacity of the persistent volume claim that is being replicated.

Replace **destination** with the name of your replication destination CR.

Replace **destination-ns** with the name of the namespace where your destination is located.

For this example, the **ServiceType** value of **LoadBalancer** is used. The load balancer service is created by the source cluster to enable your source managed cluster to transfer information to a different destination managed cluster. You can use **ClusterIP** as the service type if your source and destinations are on the same cluster, or if you have Submariner network service configured. Note the address and the name of the secret to refer to when you configure the source cluster.

The **storageClassName** and **volumeSnapshotClassName** are optional parameters. Specify the values for your environment, particularly if you are using a storage class and volume snapshot class name that are different than the default values for your environment.

c. Run the following command on the destination cluster to create the **replicationdestination** resource:

```
oc create -n <destination-ns> -f replication_destination.yaml
```

Replace **destination-ns** with the name of the namespace where your destination is located.

After the **replicationdestination** resource is created, following parameters and values are added to the resource:

Parameter	Value
.status.rsync.address	IP address of the destination cluster that is used to enable the source and destination clusters to communicate.
.status.rsync.sshKeys	Name of the SSH key file that enables secure data transfer from the source cluster to the destination cluster.

d. Run the following command to copy the value of **.status.rsync.address** to use on the source cluster:

```
ADDRESS=`oc get replicationdestination <destination> -n <destination-ns> --template=
{{.status.rsync.address}}`
echo $ADDRESS
```

Replace **destination** with the name of your replication destination custom resource.

Replace **destination-ns** with the name of the namespace where your destination is located.

The output should appear similar to the following output, which is for an Amazon Web Services environment:

```
a831264645yhrjrjyer6f9e4a02eb2-5592c0b3d94dd376.elb.us-east-1.amazonaws.com
```

e. Run the following command to copy the name of the secret and the contents of the secret that are provided as the value of **.status.rsync.sshKeys**.

```
SSHKEYS=`oc get replicationdestination <destination> -n <destination-ns> --template=
{{.status.rsync.sshKeys}}`
echo $SSHKEYS
```

Replace **destination** with the name of your replication destination custom resource.

Replace **destination-ns** with the name of the namespace where your destination is located.

You will have to enter it on the source cluster when you configure the source. The output should be the name of your SSH keys secret file, which might resemble the following name:

```
volsync-rsync-dst-src-destination-name
```

2. Identify the source persistent volume claim that you want to replicate.
Note: The source persistent volume claim must be on a CSI storage class.

3. Create the **ReplicationSource** items.

a. Copy the following YAML content to create a new file called **replication_source.yaml** on the source cluster:

```
apiVersion: volsync.backube/v1alpha1
kind: ReplicationSource
metadata:
  name: <source>
  namespace: <source-ns>
spec:
  sourcePVC: <persistent_volume_claim>
  trigger:
    schedule: "*/3 * * * *" #/*
  rsync:
    sshKeys: <mysshkeys>
    address: <my.host.com>
    copyMethod: Snapshot
    storageClassName: gp2-csi
    volumeSnapshotClassName: gp2-csi
```

Replace **source** with the name for your replication source custom resource. See step *3-vi* of this procedure for instructions on how to replace this automatically.

Replace **source-ns** with the namespace of the persistent volume claim where your source is located. See step *3-vi* of this procedure for instructions on how to replace this automatically.

Replace **persistent_volume_claim** with the name of your source persistent volume claim.

Replace **mysshkeys** with the keys that you copied from the **.status.rsync.sshKeys** field of the **ReplicationDestination** when you configured it.

Replace **my.host.com** with the host address that you copied from the **.status.rsync.address** field of the **ReplicationDestination** when you configured it.

If your storage driver supports cloning, using **Clone** as the value for **copyMethod** might be a more streamlined process for the replication.

StorageClassName and **volumeSnapshotClassName** are optional parameters. If you are using a storage class and volume snapshot class name that are different than the defaults for your environment, specify those values.

You can now set up the synchronization method of the persistent volume.

b. Copy the SSH secret from the destination cluster by entering the following command against the destination cluster:

```
oc get secret -n <destination-ns> $SSHKEYS -o yaml > /tmp/secret.yaml
```

Replace **destination-ns** with the namespace of the persistent volume claim where your destination is located.

c. Open the secret file in the **vi** editor by entering the following command:

```
vi /tmp/secret.yaml
```

d. In the open secret file on the destination cluster, make the following changes:

- Change the namespace to the namespace of your source cluster. For this example, it is **source-ns**.
- Remove the owner references (**.metadata.ownerReferences**).

e. On the source cluster, create the secret file by entering the following command on the source cluster:

```
kubectl create -f /tmp/secret.yaml
```

f. On the source cluster, modify the **replication_source.yaml** file by replacing the value of the **address** and **sshKeys** in the **ReplicationSource** object with the values that you noted from the destination cluster by entering the following commands:

```
sed -i "s/<my.host.com>/$ADDRESS/g" replication_source.yaml
sed -i "s/<mysshkeys>/$SSHKEYS/g" replication_source.yaml
oc create -n <source> -f replication_source.yaml
```

Replace **my.host.com** with the host address that you copied from the **.status.rsync.address** field of the **ReplicationDestination** when you configured it.

Replace **mysshkeys** with the keys that you copied from the **.status.rsync.sshKeys** field of the **ReplicationDestination** when you configured it.

Replace **source** with the name of the persistent volume claim where your source is located.

Note: You must create the the file in the same namespace as the persistent volume claim that you want to replicate.

g. Verify that the replication completed by running the following command on the **ReplicationSource** object:

```
kubectl describe ReplicationSource -n <source-ns> <source>
```

Replace **source-ns** with the namespace of the persistent volume claim where your source is located.

Replace **source** with the name of your replication source custom resource.

If the replication was successful, the output should be similar to the following example:

```
Status:
  Conditions:
    Last Transition Time:  2021-10-14T20:48:00Z
    Message:               Synchronization in-progress
    Reason:                SyncInProgress
    Status:                True
    Type:                  Synchronizing
    Last Transition Time:  2021-10-14T20:41:41Z
    Message:               Reconcile complete
    Reason:                ReconcileComplete
    Status:                True
    Type:                  Reconciled
  Last Sync Duration:      5m20.764642395s
  Last Sync Time:          2021-10-14T20:47:01Z
  Next Sync Time:          2021-10-14T20:48:00Z
```

If the **Last Sync Time** has no time listed, then the replication is not complete.

You have a replica of your original persistent volume claim.

1.2.1.4. Configuring a restic backup

A restic-based backup copies a restic-based backup copy of the persistent volume to a location that is specified in your **restic-config.yaml** secret file. A restic backup does not synchronize data between the clusters, but provides data backup.

Complete the following steps to configure a restic-based backup:

1. Specify a repository where your backup images are stored by creating a secret that resembles the following YAML content:

```
apiVersion: v1
kind: Secret
```

```
metadata:
  name: restic-config
type: Opaque
stringData:
  RESTIC_REPOSITORY: <my-restic-repository>
  RESTIC_PASSWORD: <my-restic-password>
  AWS_ACCESS_KEY_ID: access
  AWS_SECRET_ACCESS_KEY: password
```

Replace **my-restic-repository** with the location of the S3 bucket repository where you want to store your backup files.

Replace **my-restic-password** with the encryption key that is required to access the repository.

Replace **access** and **password** with the credentials for your provider, if required. Refer to Preparing a new repository for more information.

Important: When backing up multiple persistent volume claims to the same S3 bucket, the path to the bucket must be unique for each persistent volume claim. Each persistent volume claim is backed up with a separate **ReplicationSource**, and each requires a separate restic-config secret.

By sharing the same S3 bucket, each **ReplicationSource** has write access to the entire S3 bucket.

2. Configure your backup policy by creating a **ReplicationSource** object that resembles the following YAML content:

```
apiVersion: volsync.backube/v1alpha1
kind: ReplicationSource
metadata:
  name: mydata-backup
spec:
  sourcePVC: <source>
  trigger:
    schedule: "*/30 * * * *" #/*
  restic:
    pruneIntervalDays: 14
    repository: <restic-config>
    retain:
      hourly: 6
      daily: 5
      weekly: 4
      monthly: 2
      yearly: 1
    copyMethod: Clone
  # The StorageClass to use when creating the PiT copy (same as source PVC if omitted)
  #storageClassName: my-sc-name
  # The VSC to use if the copy method is Snapshot (default if omitted)
  #volumeSnapshotClassName: my-vsc-name
```

Replace **source** with the persistent volume claim that you are backing up.

Replace the value for **schedule** with how often to run the backup. This example has the schedule for every 30 minutes. See Scheduling your synchronization for more information.

Replace the value of **PruneIntervalDays** to the number of days that elapse between instances of repacking the data to save space. The prune operation can generate significant I/O traffic while it is running.

Replace **restic-config** with the name of the secret that you created in step 1.

Set the values for **retain** to your retention policy for the backed up images.

Best practice: Use **Clone** for the value of **CopyMethod** to ensure that a point-in-time image is saved.

For additional information about the backup options, see Backup options in the VolSync documentation.

1.2.1.4.1. Restoring a restic backup

You can restore the copied data from a restic backup into a new persistent volume claim. **Best practice:** Restore only one backup into a new persistent volume claim. To restore the restic backup, complete the following steps:

1. Create a new persistent volume claim to contain the new data similar to the following example:

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: <pvc-name>
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 3Gi
```

Replace **pvc-name** with the name of the new persistent volume claim.

2. Create a **ReplicationDestination** custom resource that resembles the following example to specify where to restore the data:

```
apiVersion: volsync.backube/v1alpha1
kind: ReplicationDestination
metadata:
  name: <destination>
spec:
  trigger:
    manual: restore-once
  restic:
    repository: <restic-repo>
    destinationPVC: <pvc-name>
    copyMethod: Direct
```

Replace **destination** with the name of your replication destination CR.

Replace **restic-repo** with the path to your repository where the source is stored.

Replace **pvc-name** with the name of the new persistent volume claim where you want to restore the data. Use an existing persistent volume claim for this, rather than provisioning a new one.

The restore process only needs to be completed once, and this example restores the most recent backup. For more information about restore options, see Restore options in the VolSync documentation.

1.2.1.5. Configuring an Rclone replication

An Rclone backup copies a single persistent volume to multiple locations by using Rclone through an intermediate object storage location, like AWS S3. It can be helpful when distributing data to multiple locations.

Complete the following steps to configure an Rclone replication:

1. Create a **ReplicationSource** custom resource that resembles the following example:

```
apiVersion: volsync.backube/v1alpha1
kind: ReplicationSource
metadata:
  name: <source>
  namespace: <source-ns>
spec:
  sourcePVC: <source-pvc>
  trigger:
    schedule: "*/6 * * * *" #\*
  rclone:
    rcloneConfigSection: <intermediate-s3-bucket>
    rcloneDestPath: <destination-bucket>
    rcloneConfig: <rclone-secret>
    copyMethod: Snapshot
    storageClassName: <my-sc-name>
    volumeSnapshotClassName: <my-vsc>
```

Replace **source-pvc** with the name for your replication source custom resource.

Replace **source-ns** with the namespace of the persistent volume claim where your source is located.

Replace **source** with the persistent volume claim that you are replicating.

Replace the value of **schedule** with how often to run the replication. This example has the schedule for every 6 minutes. This value must be within quotation marks. See Scheduling your synchronization for more information.

Replace **intermediate-s3-bucket** with the path to the configuration section of the Rclone configuration file.

Replace **destination-bucket** with the path to the object bucket where you want your replicated files copied.

Replace **rclone-secret** with the name of the secret that contains your Rclone configuration information.

Set the value for **copyMethod** as **Clone**, **Direct**, or **Snapshot**. This value specifies whether the point-in-time copy is generated, and if so, what method is used for generating it.

Replace **my-sc-name** with the name of the storage class that you want to use for your point-in-time copy. If not specified, the storage class of the source volume is used.

Replace **my-vsc** with the name of the **VolumeSnapshotClass** to use, if you specified **Snapshot** as your **copyMethod**. This is not required for other types of **copyMethod**.

2. Create a **ReplicationDestination** custom resource that resembles the following example:

```
apiVersion: volsync.backube/v1alpha1
kind: ReplicationDestination
metadata:
  name: database-destination
  namespace: dest
spec:
  trigger:
    schedule: "3,9,15,21,27,33,39,45,51,57 * * * *" #\*
  rclone:
    rcloneConfigSection: <intermediate-s3-bucket>
    rcloneDestPath: <destination-bucket>
    rcloneConfig: <rclone-secret>
    copyMethod: Snapshot
    accessModes: [ReadWriteOnce]
    capacity: 10Gi
    storageClassName: <my-sc>
    volumeSnapshotClassName: <my-vsc>
```

Replace the value for **schedule** with how often to move the replication to the destination. The schedules for the source and destination must be offset to allow the data to finish replicating before it is pulled from the destination. This example has the schedule for every 6 minutes, offset by 3 minutes. This value must be within quotation marks. See Scheduling your synchronization for more information.

Replace **intermediate-s3-bucket** with the path to the configuration section of the Rclone configuration file.

Replace **destination-bucket** with the path to the object bucket where you want your replicated files copied.

Replace **rclone-secret** with the name of the secret that contains your Rclone configuration information.

Set the value for **copyMethod** as **Clone**, **Direct**, or **Snapshot**. This value specifies whether the point-in-time copy is generated, and if so, which method is used for generating it.

The value for **accessModes** specifies the access modes for the persistent volume claim. Valid values are **ReadWriteOnce** or **ReadWriteMany**.

The **capacity** specifies the size of the destination volume, which must be large enough to contain the incoming data.

Replace **my-sc** with the name of the storage class that you want to use as the destination for your point-in-time copy. If not specified, the system storage class is used.

Replace **my-vsc** with the name of the **VolumeSnapshotClass** to use, if you specified **Snapshot** as your **copyMethod**. This is not required for other types of **copyMethod**. If not included, the system default **VolumeSnapshotClass** is used.

1.2.2. Converting a replicated image to a usable persistent volume claim

You might need to use the replicated image to recover data, or create a new instance of a persistent volume claim. The copy of the image must be converted to a persistent volume claim before it can be used. To convert a replicated image to a persistent volume claim, complete the following steps:

1. When the replication is complete, identify the latest snapshot from the **ReplicationDestination** object by entering the following command:

```
$ kubectl get replicationdestination <destination> -n <destination-ns> --template=
{{.status.latestImage.name}}
```

   Note the value of the latest snapshot for when you create your persistent volume claim.

   Replace **destination** with the name of your replication destination.

   Replace **destination-ns** with the namespace of your destination.

2. Create a **pvc.yaml** file that resembles the following example:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: <pvc-name>
  namespace: <destination-ns>
spec:
  accessModes:
    - ReadWriteOnce
  dataSource:
    kind: VolumeSnapshot
    apiGroup: snapshot.storage.k8s.io
    name: <snapshot_to_replace>
  resources:
    requests:
      storage: 2Gi
```

   Replace **pvc-name** with a name for your new persistent volume claim.

   Replace **destination-ns** with the namespace where the persistent volume claim is located.

   Replace **snapshot_to_replace** with the **VolumeSnapshot** name that you found in the previous step.

   Best practice: You can update **resources.requests.storage** with a different value when the value is at least the same size as the initial source persistent volume claim.

3. Validate that your persistent volume claim is running in the environment by entering the following command:

```
$ kubectl get pvc -n <destination-ns>
```

Your original backup image is running as the main persistent volume claim.

1.2.3. Scheduling your synchronization

Select from three options when determining how you start your replications: always running, on a schedule, or manually. Scheduling your replications is an option that is often selected.

The **Schedule** option runs replications at scheduled times. A schedule is defined by a **cronspec**, so the schedule can be configured as intervals of time or as specific times. The order of the schedule values are:

"minute (0-59) hour (0-23) day-of-month (1-31) month (1-12) day-of-week (0-6)"

The replication starts when the scheduled time occurs. Your setting for this replication option might resemble the following content:

```
spec:
  trigger:
    schedule: "*/6 * * * *"
```

After enabling one of these methods, your synchronization schedule runs according to the method that you configured.

See the VolSync documentation for additional information and options.

1.3. ENABLING KLUSTERLET ADD-ONS ON CLUSTERS FROM THE MULTICLUSTER ENGINE FOR KUBERNETES OPERATOR

After you install Red Hat Advanced Cluster Management for Kubernetes and then create or import clusters with the multicluster engine for Kubernetes operator, you can enable the klusterlet add-ons for those managed clusters.

The klusterlet add-ons are not enabled by default if you created or imported clusters with the multicluster engine for Kubernetes operator. Additionally, klusterlet add-ons are not enabled by default after Red Hat Advanced Cluster Management is installed.

See the following available klusterlet add-ons:

- application-manager
- cert-policy-controller
- config-policy-controller
- iam-policy-controller
- governance-policy-framework
- search-collector

Complete the following steps to enable the klusterlet add-ons for the managed clusters after the Red Hat Advanced Cluster Management is installed:

1. Create a YAML file that is similar to the following **KlusterletAddonConfig**, with the **spec** value that represents the add-ons:

```
apiVersion: agent.open-cluster-management.io/v1
kind: KlusterletAddonConfig
metadata:
  name: <cluster_name>
  namespace: <cluster_name>
spec:
  applicationManager:
```

```
    enabled: true
  certPolicyController:
    enabled: true
  iamPolicyController:
    enabled: true
  policyController:
    enabled: true
  searchCollector:
    enabled: true
```

Note: The **policy-controller** add-on is divided into two add-ons: The **governance-policy-framework** and the **config-policy-controller**. As a result, the **policyController** controls the **governance-policy-framework** and the **config-policy-controller managedClusterAddons**.

2. Save the file as **klusterlet-addon-config.yaml**.

3. Apply the YAML by running the following command on the hub cluster:

```
oc apply -f klusterlet-addon-config.yaml
```

4. To verify whether the enabled **managedClusterAddons** are created after the **KlusterletAddonConfig** is created, run the following command:

```
oc get managedclusteraddons -n <cluster namespace>
```