Red Hat Advanced Cluster Management for Kubernetes 2.12

Observability

Observability

Last Updated: 2024-11-05

Red Hat Advanced Cluster Management for Kubernetes 2.12 Observability

Observability

Legal Notice

Abstract

Enable the observability component to gain insight about your managed clusters.

Table of Contents

CHAPTER 1. OBSERVABILITY SERVICE

Observability can help you identify and assess performance problems without additional tests and support. The Red Hat Advanced Cluster Management for Kubernetes observability component is a service you can use to understand the health and utilization of clusters, and workloads across your fleet. By using the observability service, you are able to automate and manage the components that are within observability.

Observability service uses existing and widely-adopted observability tools from the open source community. By default, multicluster observability operator is enabled during the installation of Red Hat Advanced Cluster Management. Thanos is deployed within the hub cluster for long-term metrics storage. The **observability-endpoint-operator** is automatically deployed to each imported or created managed cluster. This controller starts a metrics collector that collects the data from Red Hat OpenShift Container Platform Prometheus, then sends the data to the Red Hat Advanced Cluster Management hub cluster.

Read the following documentation for more details about the observability component:

- Observability architecture
- Observability configuration
- Enabling the observability service
- Customizing observability
- Using observability
- Managing alerts
- Search in the console
- Using observability with Red Hat Insights

1.1. OBSERVABILITY ARCHITECTURE

The **multiclusterhub-operator** enables the **multicluster-observability-operator** pod by default. You must configure the **multicluster-observability-operator** pod.

- Observability open source components
- Observability architecture diagram
- Persistent stores used in the observability service

1.1.1. Observability open source components

Observability service uses open source observability tools from community. View the following descriptions of the tools that are apart of the product observability service:

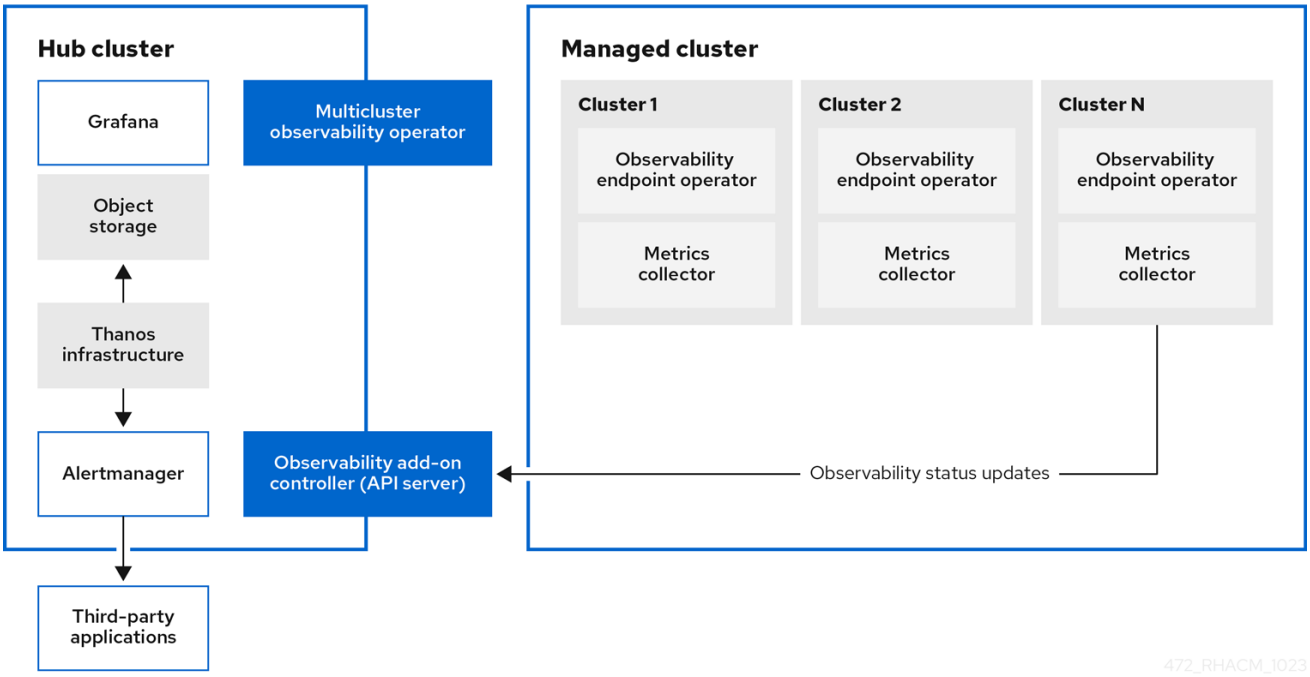
- **Thanos:** A toolkit of components that you can use to perform global querying across multiple Prometheus instances. For long-term storage of Prometheus data, persist it in any S3 compatible storage. You can also compose a highly-available and scalable metrics system.

- **Prometheus:** A monitoring and alerting tool that you can use to collect metrics from your application and store these metrics as time-series data. Store all scraped samples locally, run rules to aggregate and record new time series from existing data, and generate alerts.

- **Alertmanager:** A tool to manage and receive alerts from Prometheus. Deduplicate, group, and route alerts to your integrations such as email, Slack, and PagerDuty. Configure Alertmanager to silence and inhibit specific alerts.

1.1.2. Observability architecture diagram

The following diagram shows the components of observability:



The components of the observability architecture include the following items:

- The multicluster hub operator, also known as the **multiclusterhub-operator** pod, deploys the **multicluster-observability-operator** pod. It sends hub cluster data to your managed clusters.

- The *observability add-on controller* is the API server that automatically updates the log of the managed cluster.

- The Thanos infrastructure includes the Thanos Compactor, which is deployed by the **multicluster-observability-operator** pod. The Thanos Compactor ensures that queries are performing well by using the retention configuration, and compaction of the data in storage. To help identify when the Thanos Compactor is experiencing issues, use the four default alerts that are monitoring its health. Read the following table of default alerts:

Table 1.1. Table of default Thanos alerts

Alert	Severity	Description
ACMThanosCompactHalted	critical	An alert is sent when the compactor stops.

Alert	Severity	Description
ACMThanosCompactHighCompactionFailures	warning	An alert is sent when the compaction failure rate is greater than 5 percent.
ACMThanosCompactBucketHighOperationFailures	warning	An alert is sent when the bucket operation failure rate is greater than 5%.
ACMThanosCompactHasNotRun	warning	An alert is sent when the compactor has not uploaded anything in last 24 hours.

- The observability component deploys an instance of *Grafana* to enable data visualization with dashboards (static) or data exploration. Red Hat Advanced Cluster Management supports version 8.5.20 of Grafana. You can also design your Grafana dashboard. For more information, see *Designing your Grafana dashboard*.
- The *Prometheus Alertmanager* enables alerts to be forwarded with third-party applications. You can customize the observability service by creating custom recording rules or alerting rules. Red Hat Advanced Cluster Management supports version 0.25 of Prometheus Alertmanager.

1.1.3. Persistent stores used in the observability service

Important: Do not use the local storage operator or a storage class that uses local volumes for persistent storage. You can lose data if the pod relaunched on a different node after a restart. When this happens, the pod can no longer access the local storage on the node. Be sure that you can access the persistent volumes of the **receive** and **rules** pods to avoid data loss.

When you install Red Hat Advanced Cluster Management the following persistent volumes (PV) must be created so that Persistent Volume Claims (PVC) can attach to it automatically. As a reminder, you must define a storage class in the **MultiClusterObservability** custom resource when there is no default storage class specified or you want to use a non-default storage class to host the PVs. It is recommended to use Block Storage, similar to what Prometheus uses. Also each replica of **alertmanager**, **thanos-compactor**, **thanos-ruler**, **thanos-receive-default** and **thanos-store-shard** must have its own PV. View the following table:

Table 1.2. Table list of persistent volumes

Component name	Purpose
alertmanager	Alertmanager stores the **nflog** data and silenced alerts in its storage. **nflog** is an append-only log of active and resolved notifications along with the notified receiver, and a hash digest of contents that the notification identified.

observability-thanos-compactor	The compactor needs local disk space to store intermediate data for its processing, as well as bucket state cache. The required space depends on the size of the underlying blocks. The compactor must have enough space to download all of the source blocks, then build the compacted blocks on the disk. On-disk data is safe to delete between restarts and should be the first attempt to get crash-looping compactors unstuck. However, it is recommended to give the compactor persistent disks in order to effectively use bucket state cache in between restarts.
observability-thanos-rule	The thanos ruler evaluates Prometheus recording and alerting rules against a chosen query API by issuing queries at a fixed interval. Rule results are written back to the disk in the Prometheus 2.0 storage format. The amount of hours or days of data retained in this stateful set was fixed in the API version **observability.open-cluster-management.io/v1beta1**. It has been exposed as an API parameter in **observability.open-cluster-management.io/v1beta2**: ***RetentionInLocal***
observability-thanos-receive-default	Thanos receiver accepts incoming data (Prometheus remote-write requests) and writes these into a local instance of the Prometheus TSDB. Periodically (every 2 hours), TSDB blocks are uploaded to the object storage for long term storage and compaction. The amount of hours or days of data retained in this stateful set, which acts a local cache was fixed in API Version **observability.open-cluster-management.io/v1beta**. It has been exposed as an API parameter in **observability.open-cluster-management.io/v1beta2**: ***RetentionInLocal***
observability-thanos-store-shard	It acts primarily as an API gateway and therefore does not need a significant amount of local disk space. It joins a Thanos cluster on startup and advertises the data it can access. It keeps a small amount of information about all remote blocks on local disk and keeps it in sync with the bucket. This data is generally safe to delete across restarts at the cost of increased startup times.

Note: The time series historical data is stored in object stores. Thanos uses object storage as the primary storage for metrics and metadata related to them. For more details about the object storage and downsampling, see *Enabling observability service*.

1.1.4. Additional resources

To learn more about observability and the integrated components, see the following topics:

- See Observability service
- See Observability configuration
- See Enabling the observability service
- See the Thanos documentation.
- See the Prometheus Overview.
- See the Alertmanager documentation.

1.2. OBSERVABILITY CONFIGURATION

When the observability service is enabled, the hub cluster is always configured to collect and send metrics to the configured Thanos instance, regardless of whether hub self-management is enabled or not. When the hub cluster is self-managed, the **disableHubSelfManagement** parameter is set to **false**, which is the default setting. The **multiclusterhub-operator** enables the **multicluster-observability-operator** pod by default. You must configure the **multicluster-observability-operator** pod.

Metrics and alerts for the hub cluster appear in the **local-cluster** namespace. The **local-cluster** is only available if hub self-management is enabled. You can query the **local-cluster** metrics in the Grafana explorer. Continue reading to understand what metrics you can collect with the observability component, and for information about the observability pod capacity.

1.2.1. Metric types

By default, OpenShift Container Platform sends metrics to Red Hat using the Telemetry service. The **acm_managed_cluster_info** is available with Red Hat Advanced Cluster Management and is included with telemetry, but is *not* displayed on the Red Hat Advanced Cluster Management *Observe environments overview* dashboard.

View the following table of metric types that are supported by the framework:

Table 1.3. Parameter table

Metric name	Metric type	Labels/tags	Status
acm_managed_cluster_info	Gauge	**hub_cluster_id**, **managed_cluster_id**, **vendor**, **cloud**, **version**, **available**, **created_via**, **core_worker**, **socket_worker**	Stable
config_policies_evaluation_duration_seconds_bucket	Histogram	None	Stable. Read *Governance metric* for more details.
config_policies_evaluation_duration_seconds_count	Histogram	None	Stable. Refer to *Governance metric* for more details.

Metric name	Metric type	Labels/tags	Status
config_policies_evaluation_duration_seconds_sum	Histogram	None	Stable. Read *Governance metric* for more details.
policy_governance_info	Gauge	**type**, **policy**, **policy_namespace**, **cluster_namespace**	Stable. Review *Governance metric* for more details.
policyreport_info	Gauge	**managed_cluster_id**, **category**, **policy**, **result**, **severity**	Stable. Read *Managing insight _PolicyReports_* for more details.
search_api_db_connection_failed_total	Counter	None	Stable. See the *Search components* section in the *Searching in the console* documentation.
search_api_dbquery_duration_seconds	Histogram	None	Stable. See the *Search components* section in the *Searching in the console* documentation.
search_api_requests	Histogram	None	Stable. See the *Search components* section in the *Searching in the console* documentation.
search_indexer_request_count	Counter	None	Stable. See the *Search components* section in the *Searching in the console* documentation.
search_indexer_request_duration	Histogram	None	Stable. See the *Search components* section in the *Searching in the console* documentation.
search_indexer_requests_in_flight	Gauge	None	Stable. See the *Search components* section in the *Searching in the console* documentation.
search_indexer_request_size	Histogram	None	Stable. See the *Search components* section in the *Searching in the console* documentation.

1.2.2. Observability pod capacity requests

Observability components require 2701mCPU and 11972Mi memory to install the observability service. The following table is a list of the pod capacity requests for five managed clusters with **observability-addons** enabled:

Table 1.4. Observability pod capacity requests

Deployment or StatefulSet	Container name	CPU (mCPU)	Memory (Mi)	Replicas	Pod total CPU	Pod total memory
observability-alertmanager	alertmanager	4	200	3	12	600
	config-reloader	4	25	3	12	75
	alertmanager-proxy	1	20	3	3	60
observability-grafana	grafana	4	100	2	8	200
	grafana-dashboard-loader	4	50	2	8	100
observability-observatorium-api	observatorium-api	20	128	2	40	256
observability-observatorium-operator	observatorium-operator	100	100	1	10	50
observability-rbac-query-proxy	rbac-query-proxy	20	100	2	40	200
	oauth-proxy	1	20	2	2	40
observability-thanos-compact	thanos-compact	500	1024	1	100	512
observability-thanos-query	thanos-query	300	1024	2	600	2048

Deployment or StatefulSet	Container name	CPU (mCPU)	Memory (Mi)	Replicas	Pod total CPU	Pod total memory
observability-thanos-query-frontend	thanos-query-frontend	100	256	2	200	512
observability-thanos-query-frontend-memcached	memcached	45	128	3	135	384
	exporter	5	50	3	15	150
observability-thanos-receive-controller	thanos-receive-controller	4	32	1	4	32
observability-thanos-receive-default	thanos-receive	300	512	3	900	1536
observability-thanos-rule	thanos-rule	50	512	3	150	1536
	configmap-reloader	4	25	3	12	75
observability-thanos-store-memcached	memcached	45	128	3	135	384
	exporter	5	50	3	15	150
observability-thanos-store-shard	thanos-store	100	1024	3	300	3072

1.2.3. Additional resources

- For more information about enabling observability, read Enabling the observability service.
- Read Customizing observability to learn how to customize the observability service, view metrics and other data.
- Read Using Grafana dashboards.

- Learn from the OpenShift Container Platform documentation what types of metrics are collected and sent using telemetry. See Information collected by Telemetry for information.
- Refer to Governance metric for details.
- Refer to Prometheus recording rules.
- Also refer to Prometheus alerting rules.

1.3. ENABLING THE OBSERVABILITY SERVICE

When you enable the observability service on your hub cluster, the **multicluster-observability-operator** watches for new managed clusters and automatically deploys metric and alert collection services to the managed clusters. You can use metrics and configure Grafana dashboards to make cluster resource information visible, help you save cost, and prevent service disruptions.

Monitor the status of your managed clusters with the observability component, also known as the **multicluster-observability-operator** pod.

Required access: Cluster administrator, the **open-cluster-management:cluster-manager-admin** role, or S3 administrator.

- Prerequisites
- Enabling observability from the command line interface
- Creating the MultiClusterObservability custom resource
- Enabling observability from the Red Hat OpenShift Container Platform console
- Disabling observability
- Removing observability

1.3.1. Prerequisites

- You must install Red Hat Advanced Cluster Management for Kubernetes. See Installing while connected online for more information.
- You must define a storage class in the **MultiClusterObservability** custom resource, if there is no default storage class specified.
- Direct network access to the hub cluster is required. Network access to load balancers and proxies are not supported. For more information, see Networking.
- You must configure an object store to create a storage solution.
    - **Important:** When you configure your object store, ensure that you meet the encryption requirements that are necessary when sensitive data is persisted. The observability service uses Thanos supported, stable object stores. You might not be able to share an object store bucket by multiple Red Hat Advanced Cluster Management observability installations. Therefore, for each installation, provide a separate object store bucket.
    - Red Hat Advanced Cluster Management supports the following cloud providers with stable object stores:
        - Amazon Web Services S3 (AWS S3)

- Red Hat Ceph (S3 compatible API)
- Google Cloud Storage
- Azure storage
- Red Hat OpenShift Data Foundation, formerly known as Red Hat OpenShift Container Storage
- Red Hat OpenShift on IBM (ROKS)

1.3.2. Enabling observability from the command line interface

Enable the observability service by creating a **MultiClusterObservability** custom resource instance. Before you enable observability, see Observability pod capacity requests for more information.

Note:

- When observability is enabled or disabled on OpenShift Container Platform managed clusters that are managed by Red Hat Advanced Cluster Management, the observability endpoint operator updates the **cluster-monitoring-config** config map by adding additional **alertmanager** configuration that automatically restarts the local Prometheus.
- The observability endpoint operator updates the **cluster-monitoring-config** config map by adding additional **alertmanager** configurations that automatically restart the local Prometheus. When you insert the **alertmanager** configuration in the OpenShift Container Platform managed cluster, the configuration removes the settings that relate to the retention field of the Prometheus metrics.

Complete the following steps to enable the observability service:

1. Log in to your Red Hat Advanced Cluster Management hub cluster.
2. Create a namespace for the observability service with the following command:

```
oc create namespace open-cluster-management-observability
```

3. Generate your pull-secret. If Red Hat Advanced Cluster Management is installed in the **open-cluster-management** namespace, run the following command:

```
DOCKER_CONFIG_JSON=`oc extract secret/multiclusterhub-operator-pull-secret -n open-cluster-management --to=-`
```

   a. If the **multiclusterhub-operator-pull-secret** is not defined in the namespace, copy the **pull-secret** from the **openshift-config** namespace into the **open-cluster-management-observability** namespace by running the following command:

```
DOCKER_CONFIG_JSON=`oc extract secret/pull-secret -n openshift-config --to=-`
```

   b. Create the pull-secret in the **open-cluster-management-observability** namespace by running the following command:

```
oc create secret generic multiclusterhub-operator-pull-secret \
  -n open-cluster-management-observability \
  --from-literal=.dockerconfigjson="$DOCKER_CONFIG_JSON" \
```

```
--type=kubernetes.io/dockerconfigjson
```

Important: If you modify the global pull secret for your cluster by using the OpenShift Container Platform documentation, be sure to also update the global pull secret in the observability namespace. See Updating the global pull secret for more details.

4. Create a secret for your object storage for your cloud provider. Your secret must contain the credentials to your storage solution. For example, run the following command:

```
oc create -f thanos-object-storage.yaml -n open-cluster-management-observability
```

View the following examples of secrets for the supported object stores:

- For Amazon S3 or S3 compatible, your secret might resemble the following file:

```
apiVersion: v1
kind: Secret
metadata:
  name: thanos-object-storage
  namespace: open-cluster-management-observability
type: Opaque
stringData:
  thanos.yaml: |
    type: s3
    config:
      bucket: YOUR_S3_BUCKET
      endpoint: YOUR_S3_ENDPOINT 1
      insecure: true
      access_key: YOUR_ACCESS_KEY
      secret_key: YOUR_SECRET_KEY
```

1 Enter the URL without the protocol. Enter the URL for your Amazon S3 endpoint that might resemble the following URL: **s3.us-east-1.amazonaws.com**.

For more details, see the Amazon Simple Storage Service user guide .

- For Google Cloud Platform, your secret might resemble the following file:

```
apiVersion: v1
kind: Secret
metadata:
  name: thanos-object-storage
  namespace: open-cluster-management-observability
type: Opaque
stringData:
  thanos.yaml: |
    type: GCS
    config:
      bucket: YOUR_GCS_BUCKET
      service_account: YOUR_SERVICE_ACCOUNT
```

For more details, see Google Cloud Storage .

- For Azure your secret might resemble the following file:

```
apiVersion: v1
kind: Secret
metadata:
  name: thanos-object-storage
  namespace: open-cluster-management-observability
type: Opaque
stringData:
  thanos.yaml: |
    type: AZURE
    config:
      storage_account: YOUR_STORAGE_ACCT
      storage_account_key: YOUR_STORAGE_KEY
      container: YOUR_CONTAINER
      endpoint: blob.core.windows.net (1)
      max_retries: 0
```

(1) If you use the **msi_resource** path, the endpoint authentication is complete by using the system-assigned managed identity. Your value must resemble the following endpoint: **https://<storage-account-name>.blob.core.windows.net**.

If you use the **user_assigned_id** path, endpoint authentication is complete by using the user-assigned managed identity. When you use the **user_assigned_id**, the **msi_resource** endpoint default value is **https:<storage_account>.<endpoint>**. For more details, see Azure Storage documentation.

Note: If you use Azure as an object storage for a Red Hat OpenShift Container Platform cluster, the storage account associated with the cluster is not supported. You must create a new storage account.

- For Red Hat OpenShift Data Foundation, your secret might resemble the following file:

```
apiVersion: v1
kind: Secret
metadata:
  name: thanos-object-storage
  namespace: open-cluster-management-observability
type: Opaque
stringData:
  thanos.yaml: |
    type: s3
    config:
      bucket: YOUR_RH_DATA_FOUNDATION_BUCKET
      endpoint: YOUR_RH_DATA_FOUNDATION_ENDPOINT (1)
      insecure: false
      access_key: YOUR_RH_DATA_FOUNDATION_ACCESS_KEY
      secret_key: YOUR_RH_DATA_FOUNDATION_SECRET_KEY
```

(1) Enter the URL without the protocol. Enter the URL for your Red Hat OpenShift Data Foundation endpoint that might resemble the following URL: **example.redhat.com:443**.

For more details, see Red Hat OpenShift Data Foundation .

- For Red Hat OpenShift on IBM (ROKS), your secret might resemble the following file:

```
apiVersion: v1
kind: Secret
metadata:
  name: thanos-object-storage
  namespace: open-cluster-management-observability
type: Opaque
stringData:
  thanos.yaml: |
    type: s3
    config:
      bucket: YOUR_ROKS_S3_BUCKET
      endpoint: YOUR_ROKS_S3_ENDPOINT 1
      insecure: true
      access_key: YOUR_ROKS_ACCESS_KEY
      secret_key: YOUR_ROKS_SECRET_KEY
```

1 Enter the URL without the protocol. Enter the URL for your Red Hat OpenShift Data Foundation endpoint that might resemble the following URL: **example.redhat.com:443**.

For more details, follow the IBM Cloud documentation, Cloud Object Storage. Be sure to use the service credentials to connect with the object storage. For more details, follow the IBM Cloud documentation, Cloud Object Store and Service Credentials.

1.3.2.1. Configuring storage for AWS Security Token Service

For Amazon S3 or S3 compatible storage, you can also use short term, limited-privilege credentials that are generated with AWS Security Token Service (AWS STS). Refer to AWS Security Token Service documentation for more details.

Generating access keys using AWS Security Service require the following additional steps:

1. Create an IAM policy that limits access to an S3 bucket.
2. Create an IAM role with a trust policy to generate JWT tokens for OpenShift Container Platform service accounts.
3. Specify annotations for the observability service accounts that requires access to the S3 bucket. You can find an example of how observability on Red Hat OpenShift Service on AWS (ROSA) cluster can be configured to work with AWS STS tokens in the *Set environment* step. See Red Hat OpenShift Service on AWS (ROSA) for more details, along with ROSA with STS explained for an in-depth description of the requirements and setup to use STS tokens.

1.3.2.2. Generating access keys using the AWS Security Service

Complete the following steps to generate access keys using the AWS Security Service:

1. Set up the AWS environment. Run the following commands:

```
export POLICY_VERSION=$(date +"%m-%d-%y")
export TRUST_POLICY_VERSION=$(date +"%m-%d-%y")
export CLUSTER_NAME=<my-cluster>
export S3_BUCKET=$CLUSTER_NAME-acm-observability
export REGION=us-east-2
export NAMESPACE=open-cluster-management-observability
```

```
export SA=tbd
export SCRATCH_DIR=/tmp/scratch
export OIDC_PROVIDER=$(oc get authentication.config.openshift.io cluster -o json | jq -r
.spec.serviceAccountIssuer| sed -e "s/^https:\/\///")
export AWS_ACCOUNT_ID=$(aws sts get-caller-identity --query Account --output text)
export AWS_PAGER=""
rm -rf $SCRATCH_DIR
mkdir -p $SCRATCH_DIR
```

2. Create an S3 bucket with the following command:

```
aws s3 mb s3://$S3_BUCKET
```

3. Create a **s3-policy** JSON file for access to your S3 bucket. Run the following command:

```
{
    "Version": "$POLICY_VERSION",
    "Statement": [
        {
            "Sid": "Statement",
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket",
                "s3:GetObject",
                "s3:DeleteObject",
                "s3:PutObject",
                "s3:PutObjectAcl",
                "s3:CreateBucket",
                "s3:DeleteBucket"
            ],
            "Resource": [
                "arn:aws:s3:::$S3_BUCKET/*",
                "arn:aws:s3:::$S3_BUCKET"
            ]
        }
    ]
}
```

4. Apply the policy with the following command:

```
S3_POLICY=$(aws iam create-policy --policy-name $CLUSTER_NAME-acm-obs \
--policy-document file://$SCRATCH_DIR/s3-policy.json \
--query 'Policy.Arn' --output text)
echo $S3_POLICY
```

5. Create a **TrustPolicy** JSON file. Run the following command:

```
{
 "Version": "$TRUST_POLICY_VERSION",
 "Statement": [
   {
     "Effect": "Allow",
     "Principal": {
       "Federated": "arn:aws:iam::${AWS_ACCOUNT_ID}:oidc-provider/${OIDC_PROVIDER}"
```

```
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "${OIDC_PROVIDER}:sub": [
            "system:serviceaccount:${NAMESPACE}:observability-thanos-query",
            "system:serviceaccount:${NAMESPACE}:observability-thanos-store-shard",
            "system:serviceaccount:${NAMESPACE}:observability-thanos-compact"
            "system:serviceaccount:${NAMESPACE}:observability-thanos-rule",
            "system:serviceaccount:${NAMESPACE}:observability-thanos-receive",
          ]
        }
      }
    }
  ]
}
```

6. Create a role for AWS Prometheus and CloudWatch with the following command:

```
S3_ROLE=$(aws iam create-role \
  --role-name "$CLUSTER_NAME-acm-obs-s3" \
  --assume-role-policy-document file://$SCRATCH_DIR/TrustPolicy.json \
  --query "Role.Arn" --output text)
echo $S3_ROLE
```

7. Attach the policies to the role. Run the following command:

```
aws iam attach-role-policy \
  --role-name "$CLUSTER_NAME-acm-obs-s3" \
  --policy-arn $S3_POLICY
```

Your secret might resemble the following file. The **config** section specifies **signature_version2: false** and does not specify **access_key** and **secret_key**:

```
apiVersion: v1
kind: Secret
metadata:
  name: thanos-object-storage
  namespace: open-cluster-management-observability
type: Opaque
stringData:
  thanos.yaml: |
 type: s3
 config:
   bucket: $S3_BUCKET
   endpoint: s3.$REGION.amazonaws.com
   signature_version2: false
```

8. Specify the service account annotations in the **MultiClusterObservability** custom resource as described in *Creating the MultiClusterObservability custom resource* section.

9. Retrieve the S3 access key and secret key for your cloud providers with the following commands. You must decode, edit, and encode your **base64** string in the secret:

   a. To edit and decode the S3 access key for your cloud provider, run the following command:

```
YOUR_CLOUD_PROVIDER_ACCESS_KEY=$(oc -n open-cluster-management-observability get secret <object-storage-secret> -o jsonpath="{.data.thanos\.yaml}" | base64 --decode | grep access_key | awk '{print $2}')
```

b. To view the access key for your cloud provider, run the following command:

```
echo $YOUR_CLOUD_PROVIDER_ACCESS_KEY
```

c. To edit and decode the secret key for your cloud provider, run the following command:

```
YOUR_CLOUD_PROVIDER_SECRET_KEY=$(oc -n open-cluster-management-observability get secret <object-storage-secret> -o jsonpath="{.data.thanos\.yaml}" | base64 --decode | grep secret_key | awk '{print $2}')
```

d. Run the following command to view the secret key for your cloud provider:

```
echo $YOUR_CLOUD_PROVIDER_SECRET_KEY
```

10. Verify that observability is enabled by checking the pods for the following deployments and stateful sets. You might receive the following information:

```
observability-thanos-query (deployment)
observability-thanos-compact (statefulset)
observability-thanos-receive-default  (statefulset)
observability-thanos-rule   (statefulset)
observability-thanos-store-shard-x  (statefulsets)
```

1.3.2.3. Creating the MultiClusterObservability custom resource

Use the **MultiClusterObservability** custom resource to specify the persistent volume storage size for various components. You must set the storage size during the initial creation of the **MultiClusterObservability** custom resource. When you update the storage size values post-deployment, changes take effect only if the storage class supports dynamic volume expansion. For more information, see Expanding persistent volumes from the Red Hat OpenShift Container Platform documentation.

Complete the following steps to create the **MultiClusterObservability** custom resource on your hub cluster:

1. Create the **MultiClusterObservability** custom resource YAML file named ***multiclusterobservability_cr.yaml***.
   View the following default YAML file for observability:

```
apiVersion: observability.open-cluster-management.io/v1beta2
kind: MultiClusterObservability
metadata:
  name: observability
spec:
  observabilityAddonSpec: {}
  storageConfig:
    metricObjectStorage:
      name: thanos-object-storage
      key: thanos.yaml
```

You might want to modify the value for the **retentionConfig** parameter in the **advanced** section. For more information, see Thanos Downsampling resolution and retention. Depending on the number of managed clusters, you might want to update the amount of storage for stateful sets. If your S3 bucket is configured to use STS tokens, annotate the service accounts to use STS with S3 role. View the following configuration:

```
spec:
  advanced:
    compact:
      serviceAccountAnnotations:
        eks.amazonaws.com/role-arn: $S3_ROLE
    store:
      serviceAccountAnnotations:
        eks.amazonaws.com/role-arn: $S3_ROLE
    rule:
      serviceAccountAnnotations:
        eks.amazonaws.com/role-arn: $S3_ROLE
    receive:
      serviceAccountAnnotations:
        eks.amazonaws.com/role-arn: $S3_ROLE
    query:
      serviceAccountAnnotations:
        eks.amazonaws.com/role-arn: $S3_ROLE
```

See Observability API for more information.

2. To deploy on infrastructure machine sets, you must set a label for your set by updating the **nodeSelector** in the **MultiClusterObservability** YAML. Your YAML might resemble the following content:

```
nodeSelector:
  node-role.kubernetes.io/infra: ""
```

For more information, see Creating infrastructure machine sets.

3. Apply the observability YAML to your cluster by running the following command:

```
oc apply -f multiclusterobservability_cr.yaml
```

All the pods in **open-cluster-management-observability** namespace for Thanos, Grafana and Alertmanager are created. All the managed clusters connected to the Red Hat Advanced Cluster Management hub cluster are enabled to send metrics back to the Red Hat Advanced Cluster Management Observability service.

4. Validate that the observability service is enabled and the data is populated by launching the Grafana dashboards.

5. Click the **Grafana link** that is near the console header, from either the console *Overview* page or the *Clusters* page.

6. Access the **multicluster-observability-operator** deployment to verify that the **multicluster-observability-operator** pod is being deployed by the **multiclusterhub-operator** deployment. Run the following command:

```
oc get deploy multicluster-observability-operator -n open-cluster-management --show-labels
```

You might receive the following results:

```
NAME                                READY   UP-TO-DATE   AVAILABLE   AGE   LABELS
multicluster-observability-operator   1/1     1            1           35m
installer.name=multiclusterhub,installer.namespace=open-cluster-management
```

7. View the **labels** section of the **multicluster-observability-operator** deployment for labels that are associated with the resource. The **labels** section might contain the following details:

```
labels:
  installer.name: multiclusterhub
  installer.namespace: open-cluster-management
```

8. *Optional:* If you want to exclude specific managed clusters from collecting the observability data, add the following cluster label to your clusters: **observability: disabled**.

The observability service is enabled. After you enable the observability service, the following functions are initiated:

- All the alert managers from the managed clusters are forwarded to the Red Hat Advanced Cluster Management hub cluster.
- All the managed clusters that are connected to the Red Hat Advanced Cluster Management hub cluster are enabled to send alerts back to the Red Hat Advanced Cluster Management observability service. You can configure the Red Hat Advanced Cluster Management Alertmanager to take care of deduplicating, grouping, and routing the alerts to the correct receiver integration such as email, PagerDuty, or OpsGenie. You can also handle silencing and inhibition of the alerts.
  Note: Alert forwarding to the Red Hat Advanced Cluster Management hub cluster feature is only supported by managed clusters on a supported OpenShift Container Platform version. After you install Red Hat Advanced Cluster Management with observability enabled, alerts are automatically forwarded to the hub cluster. See Forwarding alerts to learn more.

1.3.3. Enabling observability from the Red Hat OpenShift Container Platform console

Optionally, you can enable observability from the Red Hat OpenShift Container Platform console, create a project named **open-cluster-management-observability**. Complete the following steps:

1. Create an image pull-secret named, **multiclusterhub-operator-pull-secret** in the **open-cluster-management-observability** project.
2. Create your object storage secret named, **thanos-object-storage** in the **open-cluster-management-observability** project.
3. Enter the object storage secret details, then click **Create**. See step four of the *Enabling observability* section to view an example of a secret.
4. Create the **MultiClusterObservability** custom resource instance. When you receive the following message, the observability service is enabled successfully from OpenShift Container Platform: **Observability components are deployed and running**.

1.3.3.1. Verifying the Thanos version

After Thanos is deployed on your cluster, verify the Thanos version from the command line interface (CLI).

After you log in to your hub cluster, run the following command in the observability pods to receive the Thanos version:

```
thanos --version
```

The Thanos version is displayed.

1.3.4. Disabling observability

You can disable observability, which stops data collection on the Red Hat Advanced Cluster Management hub cluster.

1.3.4.1. Disabling observability on all clusters

Disable observability by removing observability components on all managed clusters. Update the **multicluster-observability-operator** resource by setting **enableMetrics** to **false**. Your updated resource might resemble the following change:

```
spec:
  imagePullPolicy: Always
  imagePullSecret: multiclusterhub-operator-pull-secret
  observabilityAddonSpec: # The ObservabilityAddonSpec defines the global settings for all managed clusters which have observability add-on enabled
    enableMetrics: false #indicates the observability addon push metrics to hub server
```

1.3.4.2. Disabling observability on a single cluster

Disable observability by removing observability components on specific managed clusters. Complete the following steps:

1. Add the **observability: disabled** label to the **managedclusters.cluster.open-cluster-management.io** custom resource.

2. From the Red Hat Advanced Cluster Management console *Clusters* page, add the **observability=disabled** label to the specified cluster.
   Note: When a managed cluster with the observability component is detached, the **metrics-collector** deployments are removed.

1.3.5. Removing observability

When you remove the **MultiClusterObservability** custom resource, you are disabling and uninstalling the observability service. From the OpenShift Container Platform console navigation, select **Operators** > **Installed Operators** > **Advanced Cluster Manager for Kubernetes**. Remove the **MultiClusterObservability** custom resource.

1.3.6. Additional resources

- Links to cloud provider documentation for object storage information:
  - Amazon Web Services S3 (AWS S3)

    - Red Hat Ceph (S3 compatible API)
    - Google Cloud Storage
    - Azure storage
    - Red Hat OpenShift Data Foundation (formerly known as Red Hat OpenShift Container Storage)
    - Red Hat OpenShift on IBM (ROKS)

- See Using observability.
- To learn more about customizing the observability service, see Customizing observability.
- For more related topics, return to the Observability service.

1.4. CUSTOMIZING OBSERVABILITY CONFIGURATION

After you enable observability, customize the observability configuration to the specific needs of your environment. Manage and view cluster fleet data that the observability service collects.

Required access: Cluster administrator

- Creating custom rules
- Adding custom metrics
- Adding *advanced* configuration for retention
- Updating the *MultiClusterObservability* custom resource replicas from the console
- Increasing and decreasing persistent volumes and persistent volume claims
- Customizing route certification
- Customizing certificates for accessing the object store
- Configuring proxy settings for observability add-ons
- Disabling proxy settings for observability add-ons

1.4.1. Creating custom rules

Create custom rules for the observability installation by adding Prometheus recording rules and alerting rules to the observability resource.

To precalculate expensive expressions, use the recording rules abilities with Prometheus to create alert conditions and send notifications based on how you want to send an alert to an external service. The results are saved as a new set of time series. View the following examples to create a custom alert rule within the **observability-thanos-rule-custom-rules** config map:

- To get a notification for when your CPU usage paases your defined value, create the following custom alert rule:

```
data:
```

```
  custom_rules.yaml: |
    groups:
      - name: cluster-health
        rules:
        - alert: ClusterCPUHealth-jb
          annotations:
            summary: Notify when CPU utilization on a cluster is greater than the defined
utilization limit
            description: "The cluster has a high CPU usage: {{ $value }} core for {{ $labels.cluster
}} {{ $labels.clusterID }}."
          expr: |
            max(cluster:cpu_usage_cores:sum) by (clusterID, cluster, prometheus) > 0
          for: 5s
          labels:
            cluster: "{{ $labels.cluster }}"
            prometheus: "{{ $labels.prometheus }}"
            severity: critical
```

Notes:

- When you update your custom rules, **observability-thanos-rule** pods restart automatically.
- You can create multiple rules in the configuration.
- The default alert rules are in the **observability-thanos-rule-default-rules** config map of the **open-cluster-management-observability** namespace.

- To create a custom recording rule to get the sum of the container memory cache of a pod, create the following custom rule:

```
data:
  custom_rules.yaml: |
    groups:
      - name: container-memory
        rules:
        - record: pod:container_memory_cache:sum
          expr: sum(container_memory_cache{pod!=""}) BY (pod, container)
```

Note: After you make changes to the config map, the configuration automatically reloads. The configuration reloads because of the **config-reload** within the **observability-thanos-rule** sidecar.

To verify that the alert rules are functioning correctly, go to the Grafana dashboard, select the **Explore** page, and query **ALERTS**. The alert is only available in Grafana if you created the alert.

1.4.2. Adding custom metrics

Add metrics to the **metrics_list.yaml** file to collect from managed clusters. Complete the following steps:

1. Before you add a custom metric, verify that **mco observability** is enabled with the following command:

```
oc get mco observability -o yaml
```

2. Check for the following message in the **status.conditions.message** section reads:

```
Observability components are deployed and running
```

3. Create the **observability-metrics-custom-allowlist** config map in the **open-cluster-management-observability** namespace with the following command:

```
oc apply -n open-cluster-management-observability -f observability-metrics-custom-allowlist.yaml
```

4. Add the name of the custom metric to the **metrics_list.yaml** parameter. Your YAML for the config map might resemble the following content:

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: observability-metrics-custom-allowlist
data:
  metrics_list.yaml: |
    names: 1
      - node_memory_MemTotal_bytes
    rules: 2
    - record: apiserver_request_duration_seconds:histogram_quantile_90
      expr:
histogram_quantile(0.90,sum(rate(apiserver_request_duration_seconds_bucket{job=\"apiserver\",
        verb!=\"WATCH\"}[5m])) by (verb,le))
```

1. **Optional:** Add the name of the custom metrics that are to be collected from the managed cluster.

2. **Optional:** Enter only one value for the **expr** and **record** parameter pair to define the query expression. The metrics are collected as the name that is defined in the **record** parameter from your managed cluster. The metric value returned are the results after you run the query expression.

You can use either one or both of the sections. For user workload metrics, see the *Adding user workload metrics* section.

Note: You can also individually customize each managed cluster in the custom metrics allowlist instead of applying it across your entire fleet. You can create the same YAML directly on your managed cluster to customize it.

5. Verify the data collection from your custom metric by querying the metric from the Grafana dashboard **Explore** page. You can also use the custom metrics in your own dashboard.

1.4.2.1. Adding user workload metrics

Collect OpenShift Container Platform user-defined metrics from workloads in OpenShift Container Platform to display the metrics from your Grafana dashboard. Complete the following steps:

1. Enable monitoring on your OpenShift Container Platform cluster. See *Enabling monitoring for user-defined projects* in the *Additional resources* section.

If you have a managed cluster with monitoring for user-defined workloads enabled, the user workloads are located in the **test** namespace and generate metrics. These metrics are collected by Prometheus from the OpenShift Container Platform user workload.

2. Add user workload metrics to the **observability-metrics-custom-allowlist** config map to collect the metrics in the **test** namespace. View the following example:

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: observability-metrics-custom-allowlist
  namespace: test
data:
  uwl_metrics_list.yaml: 1
    names: 2
      - sample_metrics
```

1. Enter the key for the config map data.

2. Enter the value of the config map data in YAML format. The **names** section includes the list of metric names, which you want to collect from the **test** namespace. After you create the config map, the observability collector collects and pushes the metrics from the target namespace to the hub cluster.

1.4.2.2. Removing default metrics

If you do not want to collect data for a specific metric from your managed cluster, remove the metric from the **observability-metrics-custom-allowlist.yaml** file. When you remove a metric, the metric data is not collected from your managed clusters. Complete the following steps to remove a default metric:

1. Verify that **mco observability** is enabled by using the following command:

```
oc get mco observability -o yaml
```

2. Add the name of the default metric to the **metrics_list.yaml** parameter with a hyphen **-** at the start of the metric name. View the following metric example:

```
-cluster_infrastructure_provider
```

3. Create the **observability-metrics-custom-allowlist** config map in the **open-cluster-management-observability** namespace with the following command:

```
oc apply -n open-cluster-management-observability -f observability-metrics-custom-allowlist.yaml
```

4. Verify that the observability service is not collecting the specific metric from your managed clusters. When you query the metric from the Grafana dashboard, the metric is not displayed.

1.4.3. Adding advanced configuration for retention

To update the retention for each observability component according to your need, add the **advanced** configuration section. Complete the following steps:

1. Edit the **MultiClusterObservability** custom resource with the following command:

```
oc edit mco observability -o yaml
```

2. Add the **advanced** section to the file. Your YAML file might resemble the following contents:

```
spec:
  advanced:
    retentionConfig:
      blockDuration: 2h
      deleteDelay: 48h
      retentionInLocal: 24h
      retentionResolutionRaw: 365d
      retentionResolution5m: 365d
      retentionResolution1h: 365d
    receive:
      resources:
        limits:
          memory: 4096Gi
      replicas: 3
```

Notes:

- For descriptions of all the parameters that can added into the **advanced** configuration, see the *Observability API* documentation.
- The default retention for all resolution levels, such as **retentionResolutionRaw**, **retentionResolution5m**, or **retentionResolution1h**, is 365 days (**365d**). You must set an explicit value for the resolution retention in your **MultiClusterObservability** **spec.advanced.retentionConfig** parameter.

3. If you upgraded from an earlier version and want to keep that version retention configuration, add the configuration previously mentioned. Complete the following steps:

   a. Go to your **MultiClusterObservability** resource by running the following command:

```
edit mco observability
```

   b. In the **spec.advanced.retentionConfig** parameter, apply the following configuration:

```
spec:
  advanced:
    retentionConfig:
      retentionResolutionRaw: 365d
      retentionResolution5m: 365d
      retentionResolution1h: 365d
```

1.4.4. Dynamic metrics for single-node OpenShift clusters

Dynamic metrics collection supports automatic metric collection based on certain conditions. By default, a single-node OpenShift cluster does not collect pod and container resource metrics. Once a single-node OpenShift cluster reaches a specific level of resource consumption, the defined granular metrics are collected dynamically. When the cluster resource consumption is consistently less than the threshold for a period of time, granular metric collection stops.

The metrics are collected dynamically based on the conditions on the managed cluster specified by a collection rule. Because these metrics are collected dynamically, the following Red Hat Advanced Cluster Management Grafana dashboards do not display any data. When a collection rule is activated and the corresponding metrics are collected, the following panels display data for the duration of the time that the collection rule is initiated:

- Kubernetes/Compute Resources/Namespace (Pods)
- Kubernetes/Compute Resources/Namespace (Workloads)
- Kubernetes/Compute Resources/Nodes (Pods)
- Kubernetes/Compute Resources/Pod
- Kubernetes/Compute Resources/Workload A collection rule includes the following conditions:
- A set of metrics to collect dynamically.
- Conditions written as a PromQL expression.
- A time interval for the collection, which must be set to **true**.
- A match expression to select clusters where the collect rule must be evaluated.

By default, collection rules are evaluated continuously on managed clusters every 30 seconds, or at a specific time interval. The lowest value between the collection interval and time interval takes precedence. Once the collection rule condition persists for the duration specified by the **for** attribute, the collection rule starts and the metrics specified by the rule are automatically collected on the managed cluster. Metrics collection stops automatically after the collection rule condition no longer exists on the managed cluster, at least 15 minutes after it starts.

The collection rules are grouped together as a parameter section named **collect_rules**, where it can be enabled or disabled as a group. Red Hat Advanced Cluster Management installation includes the collection rule group, **SNOResourceUsage** with two default collection rules: **HighCPUUsage** and **HighMemoryUsage**. The **HighCPUUsage** collection rule begins when the node CPU usage exceeds 70%. The **HighMemoryUsage** collection rule begins if the overall memory utilization of the single-node OpenShift cluster exceeds 70% of the available node memory. Currently, the previously mentioned thresholds are fixed and cannot be changed. When a collection rule begins for more than the interval specified by the **for** attribute, the system automatically starts collecting the metrics that are specified in the **dynamic_metrics** section.

View the list of dynamic metrics that from the **collect_rules** section, in the following YAML file:

```
collect_rules:
  - group: SNOResourceUsage
    annotations:
      description: >
        By default, a {sno} cluster does not collect pod and container resource metrics. Once a {sno}
cluster
        reaches a level of resource consumption, these granular metrics are collected dynamically.
        When the cluster resource consumption is consistently less than the threshold for a period of
time,
        collection of the granular metrics stops.
    selector:
      matchExpressions:
        - key: clusterType
          operator: In
```

```
        values: ["{sno}"]
    rules:
    - collect: SNOHighCPUUsage
      annotations:
        description: >
          Collects the dynamic metrics specified if the cluster cpu usage is constantly more than 70% for
2 minutes
      expr: (1 - avg(rate(node_cpu_seconds_total{mode=\"idle\"}[5m]))) * 100 > 70
      for: 2m
      dynamic_metrics:
        names:
          - container_cpu_cfs_periods_total
          - container_cpu_cfs_throttled_periods_total
          - kube_pod_container_resource_limits
          - kube_pod_container_resource_requests
          - namespace_workload_pod:kube_pod_owner:relabel
          - node_namespace_pod_container:container_cpu_usage_seconds_total:sum_irate
          - node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate
    - collect: SNOHighMemoryUsage
      annotations:
        description: >
          Collects the dynamic metrics specified if the cluster memory usage is constantly more than 70%
for 2 minutes
      expr: (1 - sum(:node_memory_MemAvailable_bytes:sum) /
sum(kube_node_status_allocatable{resource=\"memory\"})) * 100 > 70
      for: 2m
      dynamic_metrics:
        names:
          - kube_pod_container_resource_limits
          - kube_pod_container_resource_requests
          - namespace_workload_pod:kube_pod_owner:relabel
        matches:
          - __name__="container_memory_cache",container!=""
          - __name__="container_memory_rss",container!=""
          - __name__="container_memory_swap",container!=""
          - __name__="container_memory_working_set_bytes",container!=""
```

A **collect_rules.group** can be disabled in the **custom-allowlist** as shown in the following example. When a **collect_rules.group** is disabled, metrics collection reverts to the previous behavior. These metrics are collected at regularly, specified intervals:

```
collect_rules:
  - group: -SNOResourceUsage
```

The data is only displayed in Grafana when the rule is initiated.

1.4.5. Updating the *MultiClusterObservability* custom resource replicas from the console

If your workload increases, increase the number of replicas of your observability pods. Navigate to the Red Hat OpenShift Container Platform console from your hub cluster. Locate the **MultiClusterObservability** custom resource, and update the **replicas** parameter value for the component where you want to change the replicas. Your updated YAML might resemble the following content:

```
spec:
  advanced:
    receive:
      replicas: 6
```

For more information about the parameters within the **mco observability** custom resource, see the *Observability API* documentation.

1.4.6. Increasing and decreasing persistent volumes and persistent volume claims

Increase and decrease the persistent volume and persistent volume claims to change the amount of storage in your storage class. Complete the following steps:

1. To increase the size of the persistent volume, update the **MultiClusterObservability** custom resource if the storage class support expanding volumes.

2. To decrease the size of the persistent volumes remove the pods using the persistent volumes, delete the persistent volume and recreate them. You might experience data loss in the persistent volume. Complete the following steps:

   a. Pause the **MultiClusterObservability** operator by adding the annotation **mco-pause: "true"** to the **MultiClusterObservability** custom resource.

   b. Look for the stateful sets or deployments of the desired component. Change their replica count to **0**. This initiates a shutdown, which involves uploading local data when applicable to avoid data loss. For example, the Thanos **Receive** stateful set is named **observability-thanos-receive-default** and has three replicas by default. Therefore, you are looking for the following persistent volume claims:

      - **data-observability-thanos-receive-default-0**
      - **data-observability-thanos-receive-default-1**
      - **data-observability-thanos-receive-default-2**

   c. Delete the persistent volumes and persistent volume claims used by the desired component.

   d. In the **MultiClusterObservability** custom resource, edit the storage size in the configuration of the component to the desired amount in the storage size field. Prefix with the name of the component.

   e. Unpause the **MultiClusterObservability** operator by removing the previously added annotation.

   f. To initiate a reconcilation after having the operator paused, delete the **multicluster-observability-operator** and **observatorium-operator** pods. The pods are recreated and reconciled immediately.

3. Verify that persistent volume and volume claims are updated by checking the **MultiClusterObservability** custom resource.

1.4.7. Customizing route certificate

If you want to customize the OpenShift Container Platform route certification, you must add the routes in the **alt_names** section. To ensure your OpenShift Container Platform routes are accessible, add the

following information: **alertmanager.apps.<domainname>**, **observatorium-api.apps.<domainname>**, **rbac-query-proxy.apps.<domainname>**.

For more details, see *Replacing certificates for alertmanager route* in the Governance documentation.

Note: Users are responsible for certificate rotations and updates.

1.4.8. Customizing certificates for accessing the object store

You can configure secure connections with the observability object store by creating a **Secret** resource that contains the certificate authority and configuring the **MultiClusterObservability** custom resource. Complete the following steps:

1. To validate the object store connection, create the **Secret** object in the file that contains the certificate authority by using the following command:

```
oc create secret generic <tls_secret_name> --from-file=ca.crt=<path_to_file> -n open-cluster-management-observability
```

   a. Alternatively, you can apply the following YAML to create the secret:

```
apiVersion: v1
kind: Secret
metadata:
  name: <tls_secret_name>
  namespace: open-cluster-management-observability
type: Opaque
data:
  ca.crt: <base64_encoded_ca_certificate>
```

   Optional: If you want to enable mutual TLS, you need to add the **public.crt** and **private.key** keys in the previous secret.

2. Add the TLS secret details to the **metricObjectStorage** section by using the following command:

```
oc edit mco observability -o yaml
```

   Your file might resemble the following YAML:

```
metricObjectStorage:
  key: thanos.yaml
  name: thanos-object-storage
  tlsSecretName: tls-certs-secret (1)
  tlsSecretMountPath: /etc/minio/certs (2)
```

   (1) The value for **tlsSecretName** is the name of the **Secret** object that you previously created.

   (2) The **/etc/minio/certs/** path defined for the **tlsSecretMountPath** parameter specifies where the certificates are mounted in the Observability components. This path is required for the next step.

3. Update the **thanos.yaml** definition in the **thanos-object-storage** secret by adding the **http_config.tls_config** section with the certificate details. View the following example:

```
thanos.yaml: |
  type: s3
  config:
    bucket: "thanos"
    endpoint: "minio:9000"
    insecure: false (1)
    access_key: "minio"
    secret_key: "minio123"
    http_config:
      tls_config:
        ca_file: /etc/minio/certs/ca.crt (2)
        insecure_skip_verify: false
```

1. Set the **insecure** parameter to **false** to enable HTTPS.
2. The path for the **ca_file** parameter must match the **tlsSecretMountPath** from the **MultiClusterObservability** custom resource. The **ca.crt** must match the key in the **<tls_secret_name> Secret** resource.

Optional: If you want to enable mutual TLS, you need to add the **cert_file** and **key_file** keys to the **tls_config** section. See the following example:

```
thanos.yaml: |
  type: s3
  config:
    bucket: "thanos"
    endpoint: "minio:9000"
    insecure: false
    access_key: "minio"
    secret_key: "minio123"
    http_config:
      tls_config:
        ca_file: /etc/minio/certs/ca.crt (1)
        cert_file: /etc/minio/certs/public.crt
        key_file: /etc/minio/certs/private.key
        insecure_skip_verify: false
```

1. The path for **ca_file**, **cert_file**, and **key_file** must match the **tlsSecretMountPath** from the **MultiClusterObservability** custom resource. The **ca.crt**, **public.crt**, and **private.crt** must match the respective key in the **tls_secret_name> Secret** resource.

4. To verify that you can access the object store, check that the pods are deployed. Run the following command:

```
oc -n open-cluster-management-observability get pods -l app.kubernetes.io/name=thanos-store
```

1.4.9. Configuring proxy settings for observability add-ons

Configure the proxy settings to allow the communications from the managed cluster to access the hub cluster through a HTTP and HTTPS proxy server. Typically, add-ons do not need any special configuration to support HTTP and HTTPS proxy servers between a hub cluster and a managed cluster.

But if you enabled the observability add-on, you must complete the proxy configuration.

1.4.10. Prerequisite

- You have a hub cluster.
- You have enabled the proxy settings between the hub cluster and managed cluster.

Complete the following steps to configure the proxy settings for the observability add-on:

1. Go to the cluster namespace on your hub cluster.
2. Create an **AddOnDeploymentConfig** resource with the proxy settings by adding a **spec.proxyConfig** parameter. View the following YAML example:

```
apiVersion: addon.open-cluster-management.io/v1alpha1
kind: AddOnDeploymentConfig
metadata:
  name: <addon-deploy-config-name>
  namespace: <managed-cluster-name>
spec:
  agentInstallNamespace: open-cluster-managment-addon-observability
  proxyConfig:
    httpsProxy: "http://<username>:<password>@<ip>:<port>" 1
    noProxy: ".cluster.local,.svc,172.30.0.1" 2
```

1. For this field, you can specify either a HTTP proxy or a HTTPS proxy.
2. Include the IP address of the **kube-apiserver**.

3. To get the IP address, run following command on your managed cluster:

```
oc -n default describe svc kubernetes | grep IP:
```

4. Go to the **ManagedClusterAddOn** resource and update it by referencing the **AddOnDeploymentConfig** resource that you made. View the following YAML example:

```
apiVersion: addon.open-cluster-management.io/v1alpha1
kind: ManagedClusterAddOn
metadata:
  name: observability-controller
  namespace: <managed-cluster-name>
spec:
  installNamespace: open-cluster-managment-addon-observability
  configs:
  - group: addon.open-cluster-management.io
    resource: AddonDeploymentConfig
    name: <addon-deploy-config-name>
    namespace: <managed-cluster-name>
```

5. Verify the proxy settings. If you successfully configured the proxy settings, the metric collector deployed by the observability add-on agent on the managed cluster sends the data to the hub cluster. Complete the following steps:

a. Go to the hub cluster then the managed cluster on the Grafana dashboard.

b. View the metrics for the proxy settings.

1.4.11. Disabling proxy settings for observability add-ons

If your development needs change, you might need to disable the proxy setting for the observability add-ons you configured for the hub cluster and managed cluster. You can disable the proxy settings for the observability add-on at any time. Complete the following steps:

1. Go to the **ManagedClusterAddOn** resource.
2. Remove the referenced **AddOnDeploymentConfig** resource.

1.4.12. Customizing the managed cluster Observatorium API and Alertmanager URLs (Technology Preview)

You can customize the Observatorium API and Alertmanager URLs that the managed cluster uses to communicate with the hub cluster to maintain all Red Hat Advanced Cluster Management functions when you use a load balancer or reserve proxy. To customize the URLs, complete the following steps:

1. Add your URLs to the **advanced** section of the **MultiClusterObservability spec**. See the following example:

```
spec:
  advanced:
    customObservabilityHubURL: <yourURL>
    customAlertmanagerHubURL: <yourURL>
```

Notes:

- Only HTTPS URLs are supported. If you do not add **https://** to your URL, the scheme is added automatically.
- You can include the standard path for the Remote Write API, **/api/metrics/v1/default/api/v1/receive** in the **customObservabilityHubURL spec**. If you do not include the path, the Observability service automatically adds the path at runtime.
- Any intermediate component you use for the custom Observability hub cluster URL cannot use TLS termination because the component relies on MTLS authentication. The custom Alertmanager hub cluster URL supports intermediate component TLS termination by using your own existing certificate instructions.

    1. If you are using a **customObservabilityHubURL**, create a route object by using the following template. Replace **<intermediate_component_url>** with the intermediate component URL:

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: proxy-observatorium-api
  namespace: open-cluster-management-observability
spec:
  host: <intermediate_component_url>
  port:
```

```
    targetPort: public
  tls:
    insecureEdgeTerminationPolicy: None
    termination: passthrough
  to:
    kind: Service
    name: observability-observatorium-api
    weight: 100
  wildcardPolicy: None
```

1. If you are using a **customAlertmanagerHubURL**, create a route object by using the following template. Replace **<intermediate_component_url>** with the intermediate component URL:

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: alertmanager-proxy
  namespace: open-cluster-management-observability
spec:
  host: <intermediate_component_url>
  path: /api/v2
  port:
    targetPort: oauth-proxy
  tls:
    insecureEdgeTerminationPolicy: Redirect
    termination: reencrypt
  to:
    kind: Service
    name: alertmanager
    weight: 100
  wildcardPolicy: None
```

1.4.13. Configuring fine-grain RBAC (Technology Preview)

To restrict metric access to specific namespaces within the cluster, use fine-grain role-based access control (RBAC). Using fine-grain RBAC, you can allow application teams to only view the metrics for the namespaces that you give them permission to access.

You must configure metric access control on the hub cluster for the users of that hub cluster. On this hub cluster, a **ManagedCluster** custom resource represents every managed cluster. To configure RBAC and to select the allowed namespaces, use the rules and action verbs specified in the **ManagedCluster** custom resources.

For example, you have an application named, **my-awesome-app**, and this application is on two different managed clusters, **devcluster1** and **devcluster2**. Both clusters are in the **AwesomeAppNS** namespace. You have an **admin** user group named, **my-awesome-app-admins**, and you want to restrict this user group to only have access to metrics from only these two namespaces on the hub cluster.

In this example, to use fine-grain RBAC to restrict the user group access, complete the following steps:

1. Define a **ClusterRole** resource with permissions to access metrics. Your resource might resemble the following YAML:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
```

```
metadata:
 name: awesome-app-metrics-role
rules:
 - apiGroups:
    - "cluster.open-cluster-management.io"
   resources:
    - managedclusters: 1
   resourceNames: 2
    - devcluster1
    - devcluster2
   verbs: 3
    - metrics/AwesomeAppNS
```

1. Represents the parameter values for the managed clusters.
2. Represents the list of managed clusters.
3. Represents the namespace of the managed clusters.

2. Define a **ClusterRoleBinding** resource that binds the group, **my-awesome-app-admins**, with the **ClusterRole** resource for the **awesome-app-metrics-role**. Your resource might resemble the following YAML:

```
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
 name: awesome-app-metrics-role-binding
subjects:
 - kind: Group
   apiGroup: rbac.authorization.k8s.io
   name: my-awesome-app-admins
roleRef:
 apiGroup: rbac.authorization.k8s.io
 kind: ClusterRole
 name: awesome-app-metrics-role
```

After completing these steps, when the users in the **my-awesome-app-admins** log into the Grafana console, they have the following restrictions:

- Users see no data for dashboards that summarize fleet level data.
- Users can only select managed clusters and namespaces specified in the **ClusterRole** resource.

To set up different types of user access, define separate **ClusterRoles** and **ClusterRoleBindings** resources to represent the different managed clusters in the namespaces.

1.4.14. Additional resources

- Refer to Prometheus configuration for more information. For more information about recording rules and alerting rules, refer to the recording rules and alerting rules from the Prometheus documentation.
- For more information about viewing the dashboard, see Using Grafana dashboards.

- See Exporting metrics to external endpoints .
- See Enabling monitoring for user-defined projects.
- See the Observability API.
- For information about updating the certificate for the alertmanager route, see Replacing certificates for alertmanager.
- For more details about observability alerts, see Observability alerts
- To learn more about alert forwarding, see the Prometheus Alertmanager documentation.
- See Observability alerts for more information.
- For more topics about the observability service, see Observability service.
- See Management Workload Partitioning for more information.

1.5. USING OBSERVABILITY

Use the observability service to view the utilization of clusters across your fleet.

- Querying metrics using the observability API
- Exporting metrics to external endpoints
- Viewing and exploring data

1.5.1. Querying metrics using the observability API

Observability provides an external API for metrics to be queried through the OpenShift route, **rbac-query-proxy**. See the following options to get your queries for the **rbac-query-proxy** route:

- You can get the details of the route with the following command:

```
oc get route rbac-query-proxy -n open-cluster-management-observability
```

- You can also access the **rbac-query-proxy** route with your OpenShift OAuth access token. The token should be associated with a user or service account, which has permission to get namespaces. For more information, see Managing user-owned OAuth access tokens.

Complete the following steps to create **proxy-byo-cert** secrets for observability:

1. Get the default CA certificate and store the content of the key **tls.crt** in a local file. Run the following command:

```
oc -n openshift-ingress get secret router-certs-default -o jsonpath="{.data.tls\.crt}" | base64 -d
> ca.crt
```

2. Run the following command to query metrics:

```
curl --cacert ./ca.crt -H "Authorization: Bearer {TOKEN}"
https://{PROXY_ROUTE_URL}/api/v1/query?query={QUERY_EXPRESSION}
```

Note: The **QUERY_EXPRESSION** is the standard Prometheus query expression. For example, query the metrics **cluster_infrastructure_provider** by replacing the URL in the previously mentioned command with the following URL: **https://{PROXY_ROUTE_URL}/api/v1/query?query=cluster_infrastructure_provider**. For more details, see Querying Prometheus.

3. Run the following command to create **proxy-byo-ca** secrets using the generated certificates:

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

4. Create **proxy-byo-cert** secrets using the generated certificates by using the following command:

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

1.5.2. Exporting metrics to external endpoints

Export metrics to external endpoints, which support the Prometheus Remote-Write specification in real time. Complete the following steps to export metrics to external endpoints:

1. Create the Kubernetes secret for an external endpoint with the access information of the external endpoint in the **open-cluster-management-observability** namespace. View the following example secret:

```
apiVersion: v1
kind: Secret
metadata:
  name: victoriametrics
  namespace: open-cluster-management-observability
type: Opaque
stringData:
  ep.yaml: |
    url: http://victoriametrics:8428/api/v1/write
    http_client_config:
      basic_auth:
        username: test
        password: test
```

The **ep.yaml** is the key of the content and is used in the **MultiClusterObservability** custom resource in next step. Currently, observability supports exporting metrics to endpoints without any security checks, with basic authentication or with **tls** enablement. View the following tables for a full list of supported parameters:

Name	Description	Schema
url *required*	URL for the external endpoint.	string
http_client_config *optional*	Advanced configuration for the HTTP client.	HttpClientConfig

HttpClientConfig

Name	Description	Schema
basic_auth *optional*	HTTP client configuration for basic authentication.	BasicAuth
tls_config *optional*	HTTP client configuration for TLS.	TLSConfig

BasicAuth

Name	Description	Schema
username *optional*	User name for basic authorization.	string
password *optional*	Password for basic authorization.	string

TLSConfig

Name	Description	Schema
secret_name *required*	Name of the secret that contains certificates.	string
ca_file_key *optional*	Key of the CA certificate in the secret (only *optional* if **insecure_skip_verify** is set to true).	string
cert_file_key *required*	Key of the client certificate in the secret.	string
key_file_key *required*	Key of the client key in the secret.	string
insecure_skip_verify *optional*	Parameter to skip the verification for target certificate.	bool

2. Add the **writeStorage** parameter to the **MultiClusterObservability** custom resource for adding a list of external endppoints that you want to export. View the following example:

```
spec:
  storageConfig:
    writeStorage: 1
```

```
  - key: ep.yaml
    name: victoriametrics
```

1 Each item contains two attributes: *name* and *key*. *Name* is the name of the Kubernetes secret that contains endpoint access information, and *key* is the key of the content in the secret. If you add more than one item to the list, then the metrics are exported to multiple external endpoints.

3. View the status of metric export after the metrics export is enabled by checking the **acm_remote_write_requests_total** metric.

   a. From the OpenShift Container Platform console of your hub cluster, navigate to the *Metrics* page by clicking **Metrics** in the *Observe* section.

   b. Then query the **acm_remote_write_requests_total** metric. The value of that metric is the total number of requests with a specific response for one external endpoint, on one observatorium API instance. The **name** label is the name for the external endpoint. The **code** label is the return code of the HTTP request for the metrics export.

1.5.3. Viewing and exploring data by using dashboards

View the data from your managed clusters by accessing Grafana from the hub cluster. You can query specific alerts and add filters for the query.

For example, to explore the *cluster_infrastructure_provider* alert from a single-node OpenShift cluster, use the following query expression: **cluster_infrastructure_provider{clusterType="SNO"}**

Note: Do not set the **ObservabilitySpec.resources.CPU.limits** parameter if observability is enabled on single node managed clusters. When you set the CPU limits, it causes the observability pod to be counted against the capacity for your managed cluster. See the reference for *Management Workload Partitioning* in the *Additional resources* section.

1.5.3.1. Viewing historical data

When you query historical data, manually set your query parameter options to control how much data is displayed from the dashboard. Complete the following steps:

1. From your hub cluster, select the **Grafana link** that is in the console header.
2. Edit your cluster dashboard by selecting **Edit Panel**.
3. From the Query front-end data source in Grafana, click the *Query* tab.
4. Select **$datasource**.
5. If you want to see more data, increase the value of the *Step* parameter section. If the *Step* parameter section is empty, it is automatically calculated.
6. Find the *Custom query parameters* field and select **max_source_resolution=auto**.
7. To verify that the data is displayed, refresh your Grafana page.

Your query data appears from the Grafana dashboard.

1.5.3.2. Viewing Red Hat Advanced Cluster Management dashboards

When you enable the Red Hat Advanced Cluster Management observability service, three dashboards become available. the following dashboard descriptions:

- *Alert Analysis*: Overview dashboard of the alerts being generated within the managed cluster fleet.
- *Clusters by Alert*: Alert dashboard where you can filter by the alert name.
- *Alerts by Cluster*: Alert dashboard where you can filter by cluster, and view real-time data for alerts that are initiated or pending within the cluster environment.

1.5.3.3. Viewing the etcd table

You can also view the etcd table from the hub cluster dashboard in Grafana to learn the stability of the etcd as a data store. Select the Grafana link from your hub cluster to view the *etcd* table data, which is collected from your hub cluster. The *Leader election changes* across managed clusters are displayed.

1.5.3.4. Viewing the Kubernetes API server dashboard

View the following options to view the Kubernetes API server dashboards:

- View the cluster fleet Kubernetes API service-level overview from the hub cluster dashboard in Grafana.
  1. Navigate to the Grafana dashboard.
  2. Access the managed dashboard menu by selecting **Kubernetes** > **Service-Level Overview** > **API Server**. The *Fleet Overview* and *Top Cluster* details are displayed.
     The total number of clusters that are exceeding or meeting the targeted *service-level objective* (SLO) value for the past seven or 30-day period, offending and non-offending clusters, and API Server Request Duration is displayed.
- View the Kubernetes API service-level overview table from the hub cluster dashboard in Grafana.
  1. Navigate to the Grafana dashboard from your hub cluster.
  2. Access the managed dashboard menu by selecting **Kubernetes** > **Service-Level Overview** > **API Server**. The *Fleet Overview* and *Top Cluster* details are displayed.
     The error budget for the past seven or 30-day period, the remaining downtime, and trend are displayed.

1.5.3.5. Viewing the OpenShift Virtualization dashboard

You can view the Red Hat OpenShift Virtualization dashboard to see comprehensive insights for each cluster with the OpenShift Virtualization operator installed. The state of the operator is displayed, which is determined by active OpenShift Virtualization alerts and the conditions of the Hyperconverged Cluster Operator. Additionally, you view the number of running virtual machines and the operator version for each cluster.

The dashboard also lists alerts affecting the health of the operator and separately includes all OpenShift Virtualization alerts, even those not impacting the health of the operator. You can filter the dashboard by cluster name, operator health alerts, health impact of alerts, and alert severity.

1.5.4. Additional resources

- For more information, see Prometheus Remote-Write specification.
- Read Enabling the observability service.
- For more topics, return to Observability service.

1.5.5. Using Grafana dashboards

Use Grafana dashboards to view hub cluster and managed cluster metrics. The data displayed in the Grafana alerts dashboard relies on **alerts** metrics, originating from managed clusters. The **alerts** metric does not affect managed clusters forwarding alerts to Red Hat Advanced Cluster Management alert manager on the hub cluster. Therefore, the metrics and alerts have distinct propagation mechanisms and follow separate code paths.

Even if you see data in the Grafana alerts dashboard, that does not guarantee that the managed cluster alerts are successfully forwarding to the Red Hat Advanced Cluster Management hub cluster alert manager. If the metrics are propagated from the managed clusters, you can view the data displayed in the Grafana alerts dashboard.

To use the Grafana dashboards for your development needs, complete the following:

- Setting up the Grafana developer instance
- Designing your Grafana dashboard
- Uninstalling the Grafana developer instance

1.5.5.1. Setting up the Grafana developer instance

You can design your Grafana dashboard by creating a **grafana-dev** instance. Be sure to use the most current **grafana-dev** instance.

Complete the following steps to set up the Grafana developer instance:

1. Clone the **open-cluster-management/multicluster-observability-operator**/ repository, so that you are able to run the scripts that are in the **tools** folder.
2. Run the **setup-grafana-dev.sh** to setup your Grafana instance. When you run the script the following resources are created: **secret/grafana-dev-config**, **deployment.apps/grafana-dev**, **service/grafana-dev**, **ingress.extensions/grafana-dev**, **persistentvolumeclaim/grafana-dev**:

```
./setup-grafana-dev.sh --deploy
secret/grafana-dev-config created
deployment.apps/grafana-dev created
service/grafana-dev created
serviceaccount/grafana-dev created
clusterrolebinding.rbac.authorization.k8s.io/open-cluster-management:grafana-crb-dev
created
route.route.openshift.io/grafana-dev created
persistentvolumeclaim/grafana-dev created
oauthclient.oauth.openshift.io/grafana-proxy-client-dev created
deployment.apps/grafana-dev patched
service/grafana-dev patched
route.route.openshift.io/grafana-dev patched
```

```
oauthclient.oauth.openshift.io/grafana-proxy-client-dev patched
clusterrolebinding.rbac.authorization.k8s.io/open-cluster-management:grafana-crb-dev
patched
```

3. Switch the user role to Grafana administrator with the **switch-to-grafana-admin.sh** script.

   a. Select the Grafana URL, **https:grafana-dev-open-cluster-management-observability.{OPENSHIFT_INGRESS_DOMAIN}**, and log in.

   b. Then run the following command to add the switched user as Grafana administrator. For example, after you log in using **kubeadmin**, run following command:

```
./switch-to-grafana-admin.sh kube:admin
User <kube:admin> switched to be grafana admin
```

The Grafana developer instance is set up.

1.5.5.1.1. Verifying Grafana version

Verify the Grafana version from the command line interface (CLI) or from the Grafana user interface.

After you log in to your hub cluster, access the **observabilty-grafana** pod terminal. Run the following command:

```
grafana-cli
```

The Grafana version that is currently deployed within the cluster environment is displayed.

Alternatively, you can navigate to the *Manage* tab in the Grafana dashboard. Scroll to the end of the page, where the version is listed.

1.5.5.2. Designing your Grafana dashboard

After you set up the Grafana instance, you can design the dashboard. Complete the following steps to refresh the Grafana console and design your dashboard:

1. From the Grafana console, create a dashboard by selecting the **Create** icon from the navigation panel. Select **Dashboard**, and then click **Add new panel**.

2. From the *New Dashboard/Edit Panel* view, navigate to the *Query* tab.

3. Configure your query by selecting **Observatorium** from the data source selector and enter a PromQL query.

4. From the Grafana dashboard header, click the **Save** icon that is in the dashboard header.

5. Add a descriptive name and click **Save**.

1.5.5.2.1. Designing your Grafana dashboard with a ConfigMap

Design your Grafana dashboard with a ConfigMap. You can use the **generate-dashboard-configmap-yaml.sh** script to generate the dashboard ConfigMap, and to save the ConfigMap locally:

```
./generate-dashboard-configmap-yaml.sh "Your Dashboard Name"
Save dashboard <your-dashboard-name> to ./your-dashboard-name.yaml
```

If you do not have permissions to run the previously mentioned script, complete the following steps:

1. Select a dashboard and click the **Dashboard settings** icon.
2. Click the **JSON Model** icon from the navigation panel.
3. Copy the dashboard JSON data and paste it in the ***data*** section.
4. Modify the ***name*** and replace ***$your-dashboard-name***. Enter a universally unique identifier (UUID) in the **uid** field in **data.$your-dashboard-name.json.$$your_dashboard_json**. You can use a program such as *uuidegen* to create a UUID. Your ConfigMap might resemble the following file:

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: $your-dashboard-name
  namespace: open-cluster-management-observability
  labels:
    grafana-custom-dashboard: "true"
data:
  $your-dashboard-name.json: |-
    $your_dashboard_json
```

Notes:

- If your dashboard is created within the **grafana-dev** instance, you can take the name of the dashboard and pass it as an argument in the script. For example, a dashboard named *Demo Dashboard* is created in the **grafana-dev** instance. From the CLI, you can run the following script:

```
./generate-dashboard-configmap-yaml.sh "Demo Dashboard"
```

After running the script, you might receive the following message:

```
Save dashboard <demo-dashboard> to ./demo-dashboard.yaml
```

- If your dashboard is not in the *General* folder, you can specify the folder name in the **annotations** section of this ConfigMap:

```
annotations:
  observability.open-cluster-management.io/dashboard-folder: Custom
```

After you complete your updates for the ConfigMap, you can install it to import the dashboard to the Grafana instance.

Verify that the YAML file is created by applying the YAML from the CLI or OpenShift Container Platform console. A ConfigMap within the **open-cluster-management-observability** namespace is created. Run the following command from the CLI:

```
oc apply -f demo-dashboard.yaml
```

From the OpenShift Container Platform console, create the ConfigMap using the **demo-dashboard.yaml** file. The dashboard is located in the *Custom* folder.

1.5.5.3. Uninstalling the Grafana developer instance

When you uninstall the instance, the related resources are also deleted. Run the following command:

```
./setup-grafana-dev.sh --clean
secret "grafana-dev-config" deleted
deployment.apps "grafana-dev" deleted
serviceaccount "grafana-dev" deleted
route.route.openshift.io "grafana-dev" deleted
persistentvolumeclaim "grafana-dev" deleted
oauthclient.oauth.openshift.io "grafana-proxy-client-dev" deleted
clusterrolebinding.rbac.authorization.k8s.io "open-cluster-management:grafana-crb-dev" deleted
```

1.5.5.4. Additional resources

- See Exporting metrics to external endpoints .
- See uuidegen for instructions to create a UUID.
- See Using managed cluster labels in Grafana for more details.
- Return to the beginning of the page Using Grafana dashboard.
- For topics, see the Observing environments introduction.

1.5.6. Using managed cluster labels in Grafana

Enable managed cluster labels to use them with Grafana dashboards. When observability is enabled in the hub cluster, the **observability-managed-cluster-label-allowlist** ConfigMap is created in the **open-cluster-management-observability** namespace. The ConfigMap contains a list of managed cluster labels maintained by the **observabilty-rbac-query-proxy** pod, to populate a list of label names to filter from within the *ACM - Cluster Overview* Grafana dashboard. By default, observability ignores a subset of labels in the **observability-managed-cluster-label-allowlist** ConfigMap.

When a cluster is imported into the managed cluster fleet or modified, the **observability-rbac-query-proxy** pod watches for any changes in reference to the managed cluster labels and automatically updates the **observability-managed-cluster-label-allowlist** ConfigMap to reflect the changes. The ConfigMap contains only unique label names, which are either included in the **ignore_labels** or **labels** list. Your **observability-managed-cluster-label-allowlist** ConfigMap might resemble the following YAML file:

```
data:
  managed_cluster.yaml: |
    ignore_labels: 1
      - clusterID
      - cluster.open-cluster-management.io/clusterset
      - feature.open-cluster-management.io/addon-application-manager
      - feature.open-cluster-management.io/addon-cert-policy-controller
      - feature.open-cluster-management.io/addon-cluster-proxy
      - feature.open-cluster-management.io/addon-config-policy-controller
      - feature.open-cluster-management.io/addon-governance-policy-framework
      - feature.open-cluster-management.io/addon-iam-policy-controller
      - feature.open-cluster-management.io/addon-observability-controller
      - feature.open-cluster-management.io/addon-search-collector
      - feature.open-cluster-management.io/addon-work-manager
```

```
    - installer.name
    - installer.namespace
    - local-cluster
    - name
  labels: 2
    - cloud
    - vendor
```

+ <1> Any label that is listed in the **ignore_labels** keylist of the ConfigMap is removed from the drop-down filter on the *ACM - Clusters Overview* Grafana dashboard. <2> The labels that are enabled are displayed in the drop-down filter on the *ACM - Clusters Overview* Grafana dashboard. The values are from the **acm_managed_cluster_labels** metric, depending on the **label** key value that is selected.

Continue reading how to use managed cluster labels in Grafana:

- Adding managed cluster labels
- Enabling managed cluster labels
- Disabling managed cluster labels

1.5.6.1. Adding managed cluster labels

When you add a managed cluster label to the **observability-managed-cluster-label-allowlist** ConfigMap, the label becomes available as a filter option in Grafana. Add a unique label to the hub cluster, or managed cluster object that is associated with the managed cluster fleet. For example, if you add the label, **department=finance** to a managed cluster, the ConfigMap is updated and might resemble the following changes:

```
data:
  managed_cluster.yaml: |
    ignore_labels:
      - clusterID
      - cluster.open-cluster-management.io/clusterset
      - feature.open-cluster-management.io/addon-application-manager
      - feature.open-cluster-management.io/addon-cert-policy-controller
      - feature.open-cluster-management.io/addon-cluster-proxy
      - feature.open-cluster-management.io/addon-config-policy-controller
      - feature.open-cluster-management.io/addon-governance-policy-framework
      - feature.open-cluster-management.io/addon-iam-policy-controller
      - feature.open-cluster-management.io/addon-observability-controller
      - feature.open-cluster-management.io/addon-search-collector
      - feature.open-cluster-management.io/addon-work-manager
      - installer.name
      - installer.namespace
      - local-cluster
      - name
    labels:
      - cloud
      - department
      - vendor
```

1.5.6.2. Enabling managed cluster labels

Enable a managed cluster label that is already disabled by removing the label from the **ignore_labels** list in the **observability-managed-cluster-label-allowlist** ConfigMap.

For example, enable the **local-cluster** and **name** labels. Your **observability-managed-cluster-label-allowlist** ConfigMap might resemble the following content:

```
data:
  managed_cluster.yaml: |
    ignore_labels:
      - clusterID
      - installer.name
      - installer.namespace
    labels:
      - cloud
      - vendor
      - local-cluster
      - name
```

The ConfigMap resyncs after 30 seconds to ensure that the cluster labels are updated. After you update the ConfigMap, check the **observability-rbac-query-proxy** pod logs in the **open-cluster-management-observability** namespace to verify where the label is listed. The following information might be displayed in the pod log:

```
enabled managedcluster labels: <label>
```

From the Grafana dashboard, verify that the label is listed as a value in the *Label* drop-down menu.

1.5.6.3. Disabling managed cluster labels

Exclude a managed cluster label from being listed in the *Label* drop-down filter. Add the label name to the **ignore_labels** list. For example, your YAML might resemble the following file if you add **local-cluster** and **name** back into the **ignore_labels** list:

```
data:
  managed_cluster.yaml: |
    ignore_labels:
      - clusterID
      - installer.name
      - installer.namespace
      - local-cluster
      - name
    labels:
      - cloud
      - vendor
```

Check the **observability-rbac-query-proxy** pod logs in the **open-cluster-management-observability** namespace to verify where the label is listed. The following information might be displayed in the pod log:

```
disabled managedcluster label: <label>
```

1.5.6.4. Additional resources

- See Using Grafana dashboards.

- Return to the beginning of the page, Using managed cluster labels in Grafana .

1.6. MANAGING ALERTS

Receive and define alerts for the observability service to be notified of hub cluster and managed cluster changes.

- Prerequisites
- Configuring Alertmanager
- Forwarding alerts
- Silencing alerts
- Suppressing alerts

1.6.1. Prerequisites

- You must enable observability on your hub cluster.
- You must have the create permission for secret resources in the **open-cluster-management-observability** namespace.
- You must have edit permission on the **MultiClusterObservability** resource.

1.6.2. Configuring Alertmanager

Integrate external messaging tools such as email, Slack, and PagerDuty to receive notifications from Alertmanager. You must override the **alertmanager-config** secret in the **open-cluster-management-observability** namespace to add integrations, and configure routes for Alertmanager. Complete the following steps to update the custom receiver rules:

1. Extract the data from the **alertmanager-config** secret. Run the following command:

```
oc -n open-cluster-management-observability get secret alertmanager-config --template='{{ index .data "alertmanager.yaml" }}' |base64 -d > alertmanager.yaml
```

2. Edit and save the **alertmanager.yaml** file configuration by running the following command:

```
oc -n open-cluster-management-observability create secret generic alertmanager-config --from-file=alertmanager.yaml --dry-run -o=yaml |  oc -n open-cluster-management-observability replace secret --filename=-
```

Your updated secret might resemble the following content:

```
global
  smtp_smarthost: 'localhost:25'
  smtp_from: 'alertmanager@example.org'
  smtp_auth_username: 'alertmanager'
  smtp_auth_password: 'password'
templates:
- '/etc/alertmanager/template/*.tmpl'
route:
```

```
group_by: ['alertname', 'cluster', 'service']
group_wait: 30s
group_interval: 5m
repeat_interval: 3h
receiver: team-X-mails
routes:
- match_re:
    service: ^(foo1|foo2|baz)$
  receiver: team-X-mails
```

Your changes are applied immediately after it is modified. For an example of Alertmanager, see prometheus/alertmanager.

1.6.2.1. Mounting secrets within the Alertmanager pods

You can create **Secret** resources with arbitrary content, which can be mounted within your **alertmanager** pods for access to authorization credentials.

To reference secrets within your Alertmanager configuration, add the **Secret** resource content within the **open-cluster-management-observability** namespace and mount the content within the **alertmanager** pods. For example, to create and mount a **tls** secret, complete the following steps:

1. To create a **tls** secret with TLS certificates, run the following command:

```
oc create secret tls tls --cert=</path/to/cert.crt> --key=</path/to/cert.key> -n open-cluster-management-observability
```

2. To mount the **tls** secret to your **MultiClusterObservability** resource, add it to the **advanced** section. Your resource might resemble the following content:

```
...
advanced:
 alertmanager:
   secrets: ['tls']
```

3. To add a reference of your **tls** secret within your Alertmanager configuration, add the path of your secret to the configuration. Your resource might resemble the following configuration:

```
tls_config:
 cert_file: '/etc/alertmanager/secrets/tls/tls.crt'
 key_file: '/etc/alertmanager/secrets/tls/tls.key'
```

4. To verify that the secrets are within your **alertmanager** pods, run the following command:

```
oc -n open-cluster-management-observability get secret alertmanager-config --template='{{ index .data "alertmanager.yaml" }}' |base64 -d > alertmanager.yaml
```

Your YAML might resemble the following contents:

```
"global":
  "http_config":
    "tls_config":
      "cert_file": "/etc/alertmanager/secrets/storyverify/tls.crt"
      "key_file": "/etc/alertmanager/secrets/storyverify/tls.key"
```

5. To save the **alertmanager.yaml** configuration in the **alertmanager-config** secret, run the following command:

```
oc -n open-cluster-management-observability create secret generic alertmanager-config --from-file=alertmanager.yaml --dry-run -o=yaml
```

6. To replace the previous secret with your new secret, run the following command:

```
oc -n open-cluster-management-observability replace secret --filename=-
```

1.6.3. Forwarding alerts

After you enable observability, alerts from your OpenShift Container Platform managed clusters are automatically sent to the hub cluster. You can use the **alertmanager-config** YAML file to configure alerts with an external notification system.

View the following example of the **alertmanager-config** YAML file:

```
global:
  slack_api_url: '<slack_webhook_url>'

route:
  receiver: 'slack-notifications'
  group_by: [alertname, datacenter, app]

receivers:
- name: 'slack-notifications'
  slack_configs:
  - channel: '#alerts'
    text: 'https://internal.myorg.net/wiki/alerts/{{ .GroupLabels.app }}/{{ .GroupLabels.alertname }}'
```

If you want to configure a proxy for alert forwarding, add the following **global** entry to the **alertmanager-config** YAML file:

```
global:
  slack_api_url: '<slack_webhook_url>'
  http_config:
    proxy_url: http://****
```

1.6.3.1. Disabling alert forwarding for managed clusters

To disable alert forwarding for managed clusters, add the following annotation to the **MultiClusterObservability** custom resource:

```
metadata:
    annotations:
      mco-disable-alerting: "true"
```

When you set the annotation, the alert forwarding configuration on the managed clusters is reverted. Any changes made to the **ocp-monitoring-config** config map in the **openshift-monitoring** namespace are also reverted. Setting the annotation ensures that the **ocp-monitoring-config** config map is no longer managed or updated by the observability operator endpoint. After you update the configuration, the Prometheus instance on your managed cluster restarts.

Important: Metrics on your managed cluster are lost if you have a Prometheus instance with a persistent volume for metrics, and the Prometheus instance restarts. Metrics from the hub cluster are not affected.

When the changes are reverted, a ConfigMap named **cluster-monitoring-reverted** is created in the **open-cluster-management-addon-observability** namespace. Any new, manually added alert forward configurations are not reverted from the ConfigMap.

Verify that the hub cluster alert manager is no longer propagating managed cluster alerts to third-party messaging tools. See the previous section, *Configuring Alertmanager*.

1.6.4. Silencing alerts

Add alerts that you do not want to receive. You can silence alerts by the alert name, match label, or time duration. After you add the alert that you want to silence, an ID is created. Your ID for your silenced alert might resemble the following string, **d839aca9-ed46-40be-84c4-dca8773671da**.

Continue reading for ways to silence alerts:

- To silence a Red Hat Advanced Cluster Management alert, you must have access to the **alertmanager-main** pod in the **open-cluster-management-observability** namespace. For example, enter the following command in the pod terminal to silence **SampleAlert**:

  ```
  amtool silence add --alertmanager.url="http://localhost:9093" --author="user" --
  comment="Silencing sample alert" alertname="SampleAlert"
  ```

- Silence an alert by using multiple match labels. The following command uses **match-label-1** and **match-label-2**:

  ```
  amtool silence add --alertmanager.url="http://localhost:9093" --author="user" --
  comment="Silencing sample alert" <match-label-1>=<match-value-1> <match-label-2>=
  <match-value-2>
  ```

- If you want to silence an alert for a specific period of time, use the **--duration** flag. Run the following command to silence the **SampleAlert** for an hour:

  ```
  amtool silence add --alertmanager.url="http://localhost:9093" --author="user" --
  comment="Silencing sample alert" --duration="1h" alertname="SampleAlert"
  ```

  You can also specify a start or end time for the silenced alert. Enter the following command to silence the **SampleAlert** at a specific start time:

  ```
  amtool silence add --alertmanager.url="http://localhost:9093" --author="user" --
  comment="Silencing sample alert" --start="2023-04-14T15:04:05-07:00"
  alertname="SampleAlert"
  ```

- To view all silenced alerts that are created, run the following command:

  ```
  amtool silence --alertmanager.url="http://localhost:9093"
  ```

- If you no longer want an alert to be silenced, end the silencing of the alert by running the following command:

  ```
  amtool silence expire --alertmanager.url="http://localhost:9093" "d839aca9-ed46-40be-84c4-
  dca8773671da"
  ```

-

- To end the silencing of all alerts, run the following command:

```
amtool silence expire --alertmanager.url="http://localhost:9093" $(amtool silence query --
alertmanager.url="http://localhost:9093" -q)
```

1.6.4.1. Migrating observability storage

If you use alert silencers, you can migrate observability storage while retaining the silencers from its earlier state. To do this, migrate your Red Hat Advanced Cluster Management observability storage by creating new **StatefulSets** and **PersistentVolumes** (PV) resources that use your chosen **StorageClass** resource.

Note: The storage for PVs is different from the object storage used to store the metrics collected from your clusters.

When you use **StatefulSets** and PVs to migrate your observability data to new storage, it stores the following data components:

- **Observatorium or Thanos:** Receives data then uploads it to object storage. Some of its components store data in PVs. For this data, the Observatorium or Thanos automatically regenerates the object storage on a startup, so there is no consequence if you lose this data.
- **Alertmanager:** Only stores silenced alerts. If you want to keep these silenced alerts, you must migrate that data to the new PV.

To migrate your observability storage, complete the following steps:

1. In the **MultiClusterObservability**, set the **.spec.storageConfig.storageClass** field to the new storage class.
2. To ensure the data of the earlier **PersistentVolumes** is retained even when you delete the **PersistentVolumeClaim**, go to all your existing **PersistentVolumes**.
3. Change the **reclaimPolicy** to **"Retain": `oc patch pv <your-pv-name> -p '{"spec": {"persistentVolumeReclaimPolicy":"Retain"}}'**.
4. **Optional:** To avoid losing data, see Migrate persistent data to another Storage Class in DG 8 Operator in OCP 4.
5. Delete both the **StatefulSet** and the **PersistentVolumeClaim** in the following **StatefulSet** cases:
   a. **alertmanager-db-observability-alertmanager-<REPLICA_NUMBER>**
   b. **data-observability-thanos-<COMPONENT_NAME>**
   c. **data-observability-thanos-receive-default**
   d. **data-observability-thanos-store-shard**
   e. **Important:** You might need to delete, then re-create, the **MultiClusterObservability** operator pod so that you can create the new **StatefulSet**.
6. Re-create a new **PersistentVolumeClaim** with the same name but the correct **StorageClass**.
7. Create a new **PersistentVolumeClaim** referring to the old **PersistentVolume**.

8. Verify that the new **StatefulSet** and **PersistentVolumes** use the new **StorageClass** that you chose.

1.6.5. Suppressing alerts

Suppress Red Hat Advanced Cluster Management alerts across your clusters globally that are less severe. Suppress alerts by defining an inhibition rule in the **alertmanager-config** in the **open-cluster-management-observability** namespace.

An inhibition rule mutes an alert when there is a set of parameter matches that match another set of existing matchers. In order for the rule to take effect, both the target and source alerts must have the same label values for the label names in the **equal** list. Your **inhibit_rules** might resemble the following:

```
global:
  resolve_timeout: 1h
inhibit_rules: (1)
  - equal:
      - namespace
    source_match: (2)
      severity: critical
    target_match_re:
      severity: warning|info
```

(1) (1) The **inhibit_rules** parameter section is defined to look for alerts in the same namespace. When a **critical** alert is initiated within a namespace and if there are any other alerts that contain the severity level **warning** or **info** in that namespace, only the **critical** alerts are routed to the Alertmanager receiver. The following alerts might be displayed when there are matches:

```
ALERTS{alertname="foo", namespace="ns-1", severity="critical"}
ALERTS{alertname="foo", namespace="ns-1", severity="warning"}
```

(2) (2) If the value of the **source_match** and **target_match_re** parameters do not match, the alert is routed to the receiver:

```
ALERTS{alertname="foo", namespace="ns-1", severity="critical"}
ALERTS{alertname="foo", namespace="ns-2", severity="warning"}
```

- To view suppressed alerts in Red Hat Advanced Cluster Management, enter the following command:

```
amtool alert --alertmanager.url="http://localhost:9093" --inhibited
```

1.6.6. Additional resources

- See Customizing observability for more details.
- For more observability topics, see Observability service.

CHAPTER 2. SEARCH IN THE CONSOLE

For Red Hat Advanced Cluster Management for Kubernetes, search provides visibility into your Kubernetes resources across all of your clusters. Search also indexes the Kubernetes resources and the relationships to other resources.

- Search components
- Search customization and configurations
- Search operations and data types

2.1. SEARCH COMPONENTS

The search architecture is composed of the following components:

Table 2.1. Search component table

Component name	Metrics	Metric type	Description
search-collector			Watches the Kubernetes resources, collects the resource metadata, computes relationships for resources across all of your managed clusters, and sends the collected data to the **search-indexer**. The **search-collector** on your managed cluster runs as a pod named, **klusterlet-addon-search**.
search-indexer Receives resource metadata from the collectors and writes to PostgreSQL database. The **search-indexer** also watches resources in the hub cluster to keep track of active managed clusters.	**search_indexer_request_duration**	Histogram	Time (seconds) the search indexer takes to process a request (from managed cluster).
	search_indexer_request_size	Histogram	Total changes (add, update, delete) in the search indexer request (from managed cluster).
	search_indexer_request_count	Counter	Total requests received by the search indexer (from managed clusters).

Component name	Metrics	Metric type	Description
	search_indexer_requests_in_flight	Gauge	Total requests the search indexer is processing at a given time.
search-api Provides access to all cluster data in the **search-indexer** through GraphQL and enforces role-based access control (RBAC).	**search_api_requests**	Histogram	Histogram of HTTP requests duration in seconds.
	search_dbquery_duration_seconds	Histogram	Latency of database requests in seconds.
	search_api_db_connection_failed_total	Counter	The total number of database connection attempts that failed.
search-postgres			Stores collected data from all managed clusters in an instance of the PostgreSQL database.

Search is configured by default on the hub cluster. When you provision or manually import a managed cluster, the **klusterlet-addon-search** is enabled. If you want to disable search on your managed cluster, see Modifying the klusterlet add-ons settings of your cluster for more information.

2.2. SEARCH CUSTOMIZATION AND CONFIGURATIONS

You can modify the default values in the **search-v2-operator** custom resource. To view details of the custom resource, run the following command:

```
oc get search search-v2-operator -o yaml
```

The search operator watches the **search-v2-operator** custom resource, reconciles the changes and updates active pods. View the following descriptions of the configurations:

- PostgreSQL database storage:
  When you install Red Hat Advanced Cluster Management, the PostgreSQL database is configured to save the PostgreSQL data in an empty directory (**emptyDir**) volume. If the empty directory size is limited, you can save the PostgreSQL data on a Persistent Volume Claim (PVC) to improve search performance. You can select a storageclass from your Red Hat Advanced Cluster Management hub cluster to back up your search data. For example, if you select the **gp2** storageclass your configuration might resemble the following example:

```
apiVersion: search.open-cluster-management.io/v1alpha1
kind: Search
metadata:
  name: search-v2-operator
  namespace: open-cluster-management
```

```
    labels:
      cluster.open-cluster-management.io/backup: ""
  spec:
    dbStorage:
      size: 10Gi
      storageClassName: gp2
```

This configuration creates a PVC named **gp2-search** and is mounted to the **search-postgres** pod. By default, the storage size is **10Gi**. You can modify the storage size. For example, **20Gi** might be sufficient for about 200 managed clusters.

- Optimize cost by tuning the pod memory or CPU requirements, replica count, and update log levels for any of the four search pods (**indexer**, **database**, **queryapi**, or **collector** pod). Update the **deployment** section of the **search-v2-operator** custom resource. There are four deployments managed by the **search-v2-operator**, which can be updated individually. Your **search-v2-operator** custom resource might resemble the following file:

```
apiVersion: search.open-cluster-management.io/v1alpha1
kind: Search
metadata:
  name: search-v2-operator
  namespace: open-cluster-management
spec:
  deployments:
    collector:
      resources: 1
        limits:
          cpu: 500m
          memory: 128Mi
        requests:
          cpu: 250m
          memory: 64Mi
    indexer:
      replicaCount: 3
    database: 2
        envVar:
          - name: POSTGRESQL_EFFECTIVE_CACHE_SIZE
            value: 1024MB
          - name: POSTGRESQL_SHARED_BUFFERS
            value: 512MB
          - name: WORK_MEM
            value: 128MB
    queryapi:
      arguments: 3
      - -v=3
```

1. You can apply resources to an **indexer**, **database**, **queryapi**, or **collector** pod.
2. You can add multiple environment variables in the **envVar** section to specify a value for each variable that you name.
3. You can control the log level verbosity for any of the previous four pods by adding the **- -v=3** argument.

See the following example where memory resources are applied to the indexer pod:

```
    indexer:
      resources:
        limits:
          memory: 5Gi
        requests:
          memory: 1Gi
```

- You can define the node placement for search pods.
  You can update the **Placement** resource of search pods by using the **nodeSelector** parameter, or the **tolerations** parameter. View the following example configuration:

```
spec:
 dbStorage:
  size: 10Gi
 deployments:
  collector: {}
  database: {}
  indexer: {}
  queryapi: {}
 nodeSelector:
  node-role.kubernetes.io/infra: ""
 tolerations:
 - effect: NoSchedule
  key: node-role.kubernetes.io/infra
  operator: Exists
```

- Specify your search query by selecting the **Advanced search** drop-down button to filter the *Column*, *Operator*, and *Value* options or add a search constraint.

2.3. SEARCH OPERATIONS AND DATA TYPES

Specify your search query by using search operations as conditions. Characters such as **>, >=, <, <=, !=** are supported. See the following search operation table:

Table 2.2. Search operation table

Default operation	Data type	Description
=	string, number	This is the default operation.
! or **!=**	string, number	This represents the *NOT* operation, which means to exclude from the search results.
<, ⇐, >, >=	number	
>	date	Dates matching the last hour, day, week, month, and year.
*	string	Partial string match.

2.4. ADDITIONAL RESOURCES

- For instruction about how to manage search, see Managing search.
- For more topics about the Red Hat Advanced Cluster Management for Kubernetes console, see Web console.

2.5. MANAGING SEARCH

Use search to query resource data from your clusters.

Required access: Cluster administrator

Continue reading the following topics:

- Creating search configurable collection
- Customizing the search console
- Querying in the console
- Updating klusterlet-addon-search deployments on managed clusters

2.5.1. Creating search configurable collection

To define which Kubernetes resources get collected from the cluster, create the **search-collector-config** config map. Complete the following steps:

1. Run the following command to create the **search-collector-config** config map:

```
oc apply -f <your-search-collector-config>.yaml
```

2. List the resources in the allow (**data.AllowedResources**) and deny list (**data.DeniedResources**) sections within the config map. Your config map might resemble the following YAML file:

```
apiVersion: v1
kind: ConfigMap
metadata:
 name: search-collector-config
 namespace: <namespace where search-collector add-on is deployed>
data:
 AllowedResources: |- 1
   - apiGroups:
       - "*"
     resources:
       - services
       - pods
   - apiGroups:
       - admission.k8s.io
       - authentication.k8s.io
     resources:
       - "*"
 DeniedResources: |- 2
   - apiGroups:
```

```
    - "*"
  resources:
    - secrets
- apiGroups:
    - admission.k8s.io
  resources:
    - policies
    - iampolicies
    - certificatepolicies
```

1. The previous config map example displays **services** and **pods** to be collected from all **apiGroups**, while allowing all resources to be collected from the **admission.k8s.io** and **authentication.k8s.io apiGroups**.

2. The config map example also prevents the central collection of **secrets** from all **apiGroups** while preventing the collection of **policies**, **iampolicies**, and **certificatepolicies** from the **apiGroup admission.k8s.io**.

Note: If you do not provide a config map, all resources are collected by default. If you only provide **AllowedResources**, all resources not listed in **AllowedResources** are automatically excluded. Resources listed in **AllowedResources** and **DeniedResources** at the same time are also excluded.

2.5.2. Customizing the search console

Customize your search results and limits. Complete the following tasks to perform the customization:

1. Customize the search result limit from the OpenShift Container Platform console.

   a. Update the **console-mce-config** in the **multicluster-engine** namespace. These settings apply to all users and might affect performance. View the following performance parameter descriptions:

      - **SAVED_SEARCH_LIMIT** - The maximum amount of saved searches for each user. By default, there is a limit of ten saved searches for each user. The default value is **10**. To update the limit, add the following key value to the **console-config** config map: **SAVED_SEARCH_LIMIT: x**.

      - **SEARCH_RESULT_LIMIT** - The maximum amount of search results displayed in the console. Default value is **1000**. To remove this limit set to **-1**.

      - **SEARCH_AUTOCOMPLETE_LIMIT** - The maximum number of suggestions retrieved for the search bar typeahead. Default value is **10,000**. To remove this limit set to **-1**.

   b. Run the following **patch** command from the OpenShift Container Platform console to change the search result to 100 items:

```
oc patch configmap console-mce-config -n multicluster-engine --type merge -p '{"data":
{"SEARCH_RESULT_LIMIT":"100"}}'
```

2. To add, edit, or remove suggested searches, create a config map named **console-search-config** and configure the **suggestedSearches** section. Suggested searches that are listed are also displayed from the console. It is required to have an **id, name, and searchText** for each search object. View the following config map example:

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: console-search-config
  namespace: <acm-namespace> 1
data:
  suggestedSearches: |-
    [
      {
        "id": "search.suggested.workloads.name",
        "name": "Workloads",
        "description": "Show workloads running on your fleet",
        "searchText": "kind:DaemonSet,Deployment,Job,StatefulSet,ReplicaSet"
      },
      {
        "id": "search.suggested.unhealthy.name",
        "name": "Unhealthy pods",
        "description": "Show pods with unhealthy status",
        "searchText": "kind:Pod
status:Pending,Error,Failed,Terminating,ImagePullBackOff,CrashLoopBackOff,RunContainerEr
ror,ContainerCreating"
      },
      {
        "id": "search.suggested.createdLastHour.name",
        "name": "Created last hour",
        "description": "Show resources created within the last hour",
        "searchText": "created:hour"
      },
      {
        "id": "search.suggested.virtualmachines.name",
        "name": "Virtual Machines",
        "description": "Show virtual machine resources",
        "searchText": "kind:VirtualMachine"
      }
    ]
```

1 Add the namespace where search is enabled.

2.5.3. Querying in the console

You can type any text value in the *Search box* and results include anything with that value from any property, such as a name or namespace. Queries that contain an empty space are not supported.

For more specific search results, include the property selector in your search. You can combine related values for the property for a more precise scope of your search. For example, search for **cluster:dev red** to receive results that match the string "red" in the **dev** cluster.

Complete the following steps to make queries with search:

1. Click **Search** in the navigation menu.
2. Type a word in the *Search box*, then Search finds your resources that contain that value.
   - As you search for resources, you receive other resources that are related to your original search result, which help you visualize how the resources interact with other resources in the

system.

- Search returns and lists each cluster with the resource that you search. For resources in the *hub* cluster, the cluster name is displayed as *local-cluster*.
- Your search results are grouped by **kind**, and each resource **kind** is grouped in a table.
- Your search options depend on your cluster objects.
- You can refine your results with specific labels. Search is case-sensitive when you query labels. See the following examples that you can select for filtering: **name**, **namespace**, **status**, and other resource fields. Auto-complete provides suggestions to refine your search. See the following example:
- Search for a single field, such as **kind:pod** to find all pod resources.
- Search for multiple fields, such as **kind:pod namespace:default** to find the pods in the default namespace.
  Notes:
  - When you search for more than one property selector with multiple values, the search returns either of the values that were queried. View the following examples:
  - When you search for **kind:Pod name:a**, any pod named **a** is returned.
  - When you search for **kind:Pod name:a,b**, any pod named **a** or **b** are returned.
  - Search for **kind:pod status:!Running** to find all pod resources where the status is not **Running**.
  - Search for **kind:pod restarts:>1** to find all pods that restarted at least twice.

3. If you want to save your search, click the **Save search** icon.
4. To download your search results, select the **Export as CSV** button.

2.5.4. Updating klusterlet-addon-search deployments on managed clusters

To collect the Kubernetes objects from the managed clusters, the **klusterlet-addon-search** pod is run on all the managed clusters where search is enabled. This deployment is run in the **open-cluster-management-agent-addon** namespace. A managed cluster with a high number of resources might require more memory for the **klusterlet-addon-search** deployment to function.

Resource requirements for the **klusterlet-addon-search** pod in a managed cluster can be specified in the **ManagedClusterAddon** custom resource in your Red Hat Advanced Cluster Management hub cluster. There is a namespace for each managed cluster with the managed cluster name. Complete the following steps:

1. Edit the **ManagedClusterAddon** custom resource from the namespace matching the managed cluster name. Run the following command to update the resource requirement in **xyz** managed cluster:

   ```
   oc edit managedclusteraddon search-collector -n xyz
   ```

2. Append the resource requirements as annotations. View the following example:

```
apiVersion: addon.open-cluster-management.io/v1alpha1
kind: ManagedClusterAddOn
metadata:
  annotations: addon.open-cluster-management.io/search_memory_limit: 2048Mi
  addon.open-cluster-management.io/search_memory_request: 512Mi
```

The annotation overrides the resource requirements on the managed clusters and automatically restarts the pod with new resource requirements.

Note: You can discover all resources defined in your managed cluster by using the API Explorer in the console. Alternatively, you can discover all resources by running the following command: **oc api-resources**

2.5.5. Additional resources

- See multicluster global hub for more details.
- See Observing environments introduction.

CHAPTER 3. USING OBSERVABILITY WITH RED HAT INSIGHTS

Red Hat Insights is integrated with Red Hat Advanced Cluster Management observability, and is enabled to help identify existing or potential problems in your clusters. Red Hat Insights helps you to identify, prioritize, and resolve stability, performance, network, and security risks. Red Hat OpenShift Container Platform offers cluster health monitoring through Red Hat OpenShift Cluster Manager. Red Hat OpenShift Cluster Manager collects anonymized, aggregated information about the health, usage, and size of the clusters. For more information, see Red Hat Insights product documentation .

When you create or import an OpenShift cluster, anonymized data from your managed cluster is automatically sent to Red Hat. This information is used to create insights, which provide cluster health information. Red Hat Advanced Cluster Management administrator can use this health information to create alerts based on severity.

Required access: Cluster administrator

3.1. PREREQUISITES

- Ensure that Red Hat Insights is enabled. For more information, see Modifying the global cluster pull secret to disable remote health reporting.
- Install OpenShift Container Platform version 4.0 or later.
- Hub cluster user, who is registered to Red Hat OpenShift Cluster Manager, must be able to manage all the Red Hat Advanced Cluster Management managed clusters in Red Hat OpenShift Cluster Manager.

3.2. MANAGING INSIGHT *POLICYREPORTS*

Red Hat Advanced Cluster Management for Kubernetes **PolicyReports** are violations that are generated by the **insights-client**. The **PolicyReports** are used to define and configure alerts that are sent to incident management systems. When there is a violation, alerts from a **PolicyReport** are sent to incident management system.

3.2.1. Searching for insight policy reports

You can search for a specific insight **PolicyReport** that has a violation, across your managed clusters. Complete the following steps:

1. Log in to your Red Hat Advanced Cluster Management hub cluster.
2. Select **Search** from the navigation menu.
3. Enter the following query: **kind:PolicyReport**.
   Note: The **PolicyReport** name matches the name of the cluster.
4. You can specify your query with the insight policy violation and categories. When you select a **PolicyReport** name, you are redirected to the *Details* page of the associated cluster. The *Insights* sidebar is automatically displayed.
5. If the search service is disabled and you want to search for an insight, run the following command from your hub cluster:

```
oc get policyreport --all-namespaces
```

3.2.2. Viewing identified issues from the console

You can view the identified issues on a specific cluster. Complete the following steps:

1. Log in to your Red Hat Advanced Cluster Management cluster.
2. Select **Overview** from the navigation menu.
3. Check the *Cluster issues* summary card. Select a severity link to view the **PolicyReports** that are associated with that severity. Details of the cluster issues and the severities are displayed from the *Search* page. Policy reports that are associated with the severity and have one or more issues appear.
4. Select a policy report to view cluster details from the *Clusters* page. The *Status* card displays information about *Nodes*, *Applications*, *Policy violations*, and *Identified issues*.
5. Select the **Number of identified issues** to view details. The *Identified issues* card represents the information from Red Hat insights. The *Identified issues* status displays the number of issues by severity. The triage levels used for the issues are the following severity categories: *Critical*, *Major*, *Low*, and *Warning*.
    a. Alternatively, you can select **Clusters** from the navigation menu.
    b. Select a managed cluster from the table to view more details.
    c. From the *Status* card, view the number of identified issues.
6. Select the number of potential issues to view the severity chart and recommended remediations for the issues from the *Potential issue* side panel. You can also use the search feature to search for recommended remediations. The remediation option displays the *Description* of the vulnerability, *Category* that vulnerability is associated with, and the *Total risk*.
7. Click the link to the vulnerability to view steps on *How to remediate* and the *Reason* for the vulnerability.
   Note: When you resolve the issue, you receive the Red Hat Insights every 30 minutes, and Red Hat Insights is updated every two hours.
8. Be sure to verify which component sent the alert message from the **PolicyReport**.
    a. Navigate to the *Governance* page and select a specific **PolicyReport**.
    b. Select the *Status* tab and click the **View details** link to view the **PolicyReport** YAML file.
    c. Locate the **source** parameter, which informs you of the component that sent the violation. The value options are **grc** and **insights**.

3.2.3. Viewing update risk predictions

View the potential risks for updating your managed clusters. Complete the following steps:

1. Log in to your managed cluster.
2. Go to the *Overview* page.

3. From the *Powered by Insights* section, you can view the percentage of clusters with predicted risks, which are listed by severity.

4. Select the number for the severity to view the list of clusters from the *Clusters* page.

5. Select the cluster that you want, then click the **Actions** drop-down button.

6. Click **Upgrade clusters** to view the risk for the upgdate.

7. From the *Upgrade clusters* modal, find the *Upgrade risks* column and click the link for the number of risks to view information in the Hybrid Cloud console.

3.3. ADDITIONAL RESOURCES

- Learn how to create custom alert rules for the **PolicyReports**, see Configuring Alertmanager for more information.

- See Observability service.