Red Hat Advanced Cluster Management for Kubernetes 2.4

Observability

Read more to learn how to optimize your managed clusters by enabling and customizing the observability service.

Last Updated: 2022-12-09

Red Hat Advanced Cluster Management for Kubernetes 2.4 Observability

Read more to learn how to optimize your managed clusters by enabling and customizing the observability service.

Legal Notice

Abstract

Read more to learn how to optimize your managed clusters by enabling and customizing the observability service.

Table of Contents

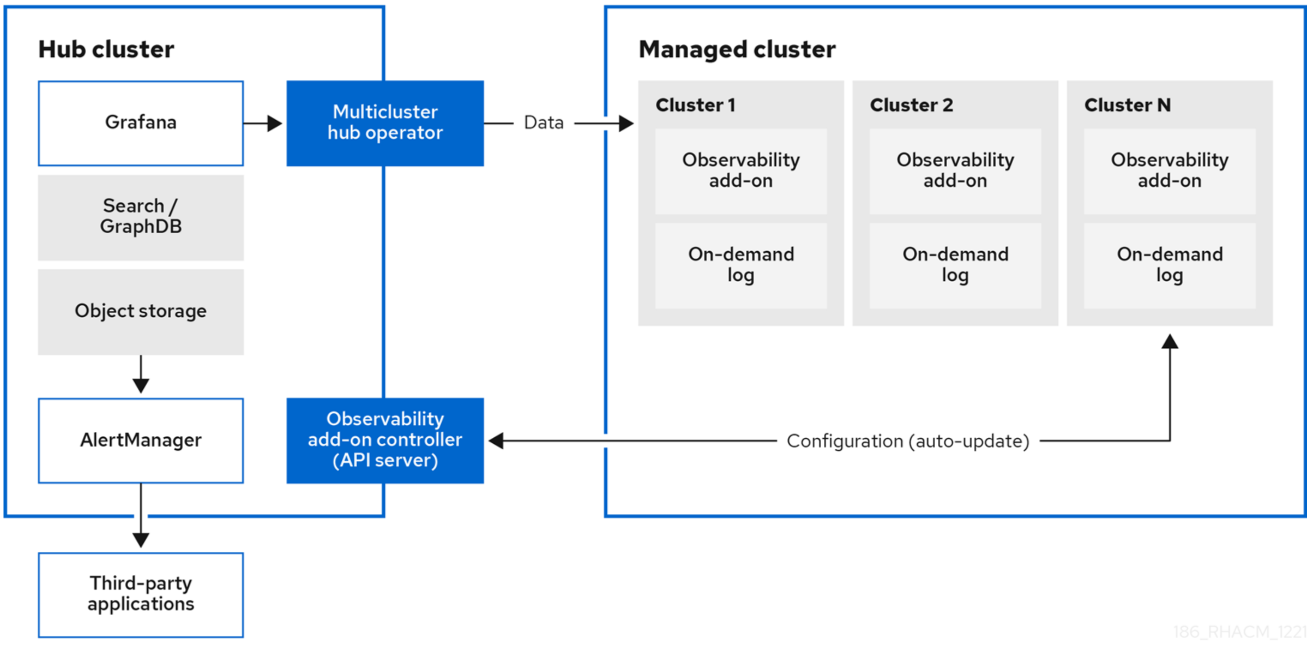
CHAPTER 1. OBSERVING ENVIRONMENTS INTRODUCTION

With the observability service enabled, you can use Red Hat Advanced Cluster Management for Kubernetes to gain insight about and optimize your managed clusters. This information can save cost and prevent unnecessary events.

- Observing environments
- Enable observability service
- Customizing observability
- Designing your Grafana dashboard
- Observability with Red Hat Insights
- Managing insight PolicyReports

1.1. OBSERVING ENVIRONMENTS

You can use Red Hat Advanced Cluster Management for Kubernetes to gain insight and optimize your managed clusters. Enable the observability service operator, **multicluster-observability-operator**, on your hub cluster to monitor the health of your managed clusters. Learn about the architecture for the multicluster observability service in the following sections.



Note: The *on-demand log* provides access for engineers to get logs for a given pod in real-time. Logs from the hub cluster are not aggregated. These logs can be accessed with the search service and other parts of the console.

- Observability service
- Metric types
- Observability pod capacity requests
- Persistent stores used in the observability service

- Support

1.1.1. Observability service

By default, observability is included with the product installation, but not enabled. Due to the requirement for persistent storage, the observability service is not enabled by default. Red Hat Advanced Cluster Management supports the following S3 compatible, stable object stores:

- Amazon S3
  Note: The object store interface in Thanos support APIs that are AWS S3 restful API compatible, or other S3 compatible object stores like Minio and Ceph.
- Google Cloud Storage
- Azure storage
- Red Hat OpenShift Data Foundation
  Important: When you configure your object store, ensure that you meet the encryption requirements necessary when sensitive data is persisted. For a complete list of the supported object stores, see Thanos documentation.

When the service is enabled, the **observability-endpoint-operator** is automatically deployed to each imported or created cluster. This controller collects the data from Red Hat OpenShift Container Platform Prometheus, then sends it to the Red Hat Advanced Cluster Management hub cluster.

If the hub cluster imports itself as the **local-cluster**, observability is also enabled on it and metrics are collected from the hub cluster.

The observability service deploys an instance of Prometheus AlertManager, which enables alerts to be forwarded with third-party applications. It also includes an instance of Grafana to enable data visualization with dashboards (static) or data exploration. Red Hat Advanced Cluster Management supports version 8.1.3 of Grafana. You can also design your Grafana dashboard. For more information, see Designing your Grafana dashboard.

You can customize the observability service by creating custom recording rules or alerting rules.

For more information about enabling observability, see Enable observability service.

1.1.2. 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.

Learn from the OpenShift Container Platform documentation what types of metrics are collected and sent using telemetry. See Information collected by Telemetry for information.

1.1.3. 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 1.1. 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	100	512	1	100	512
observability-thanos-query	thanos-query	300	1024	2	600	2048
observability-thanos-query-frontend	thanos-query-frontend	100	256	2	200	512

Deployment or StatefulSet	Container name	CPU (mCPU)	Memory (Mi)	Replicas	Pod total CPU	Pod total memory
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

1.1.4. Persistent stores used in the observability service

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** CR 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, similare 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

Persistent volume 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.
thanos-compact	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.
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***
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***
thanos-store-shard	It acts primarily as an API gateway and therefore does not need significant amounts 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 meta data related to them. For more details about the object storage and downsampling, see Enable observability service

1.1.5. Support

Red Hat Advanced Cluster Management is tested with and fully supported by Red Hat OpenShift Data Foundation (formerly Red Hat OpenShift Container Storage).

Red Hat Advanced Cluster Management supports the function of the multicluster observability operator on user-provided third-party object storage that is S3 API compatible.

Red Hat Advanced Cluster Management use commercial, reasonable efforts to assist in the identification of the root cause.

If a support ticket is raised and the root cause has been determined to be a result of the customer-provided S3 compatible object storage, then the issue must be resolved using the customer support channels.

Red Hat Advanced Cluster Management does not commit to fix support tickets raised by customers, where the root cause identified is the S3 compatible object storage provider.

See Customizing observability to learn how to configure the observability service, view metrics and other data.

1.2. ENABLE OBSERVABILITY SERVICE

Monitor the health of your managed clusters with the observability service (**multicluster-observability-operator**).

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

- Prerequisites
- Enabling observability
- Creating the MultiClusterObservability CR
- Enabling observability from the Red Hat OpenShift Container Platform console
- Using the external metric query
- Disabling observability

1.2.1. 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** CR, if there is no default storage class specified.
- You must configure an object store to create a storage solution. 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)
  Important: When you configure your object store, ensure that you meet the encryption requirements necessary when sensitive data is persisted. For more information on Thanos supported object stores, see Thanos documentation.

1.2.2. Enabling observability

Enable the observability service by creating a **MultiClusterObservability** custom resource (CR) 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 ConfigMap** by adding additional alertmanager configuration that restarts the local Prometheus automatically.

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=-`
   ```

   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. Run the following command:

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

   Then, create the pull-secret in the **open-cluster-management-observability** namespace, run 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
   ```

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 Red Hat Advanced Cluster Management, 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
      insecure: true
      access_key: YOUR_ACCESS_KEY
      secret_key: YOUR_SECRET_KEY
```

- 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
      insecure: true
      access_key: YOUR_ACCESS_KEY
      secret_key: YOUR_SECRET_KEY
```

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

- For Google, 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
      max_retries: 0
```

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
      insecure: false
      access_key: YOUR_RH_DATA_FOUNDATION_ACCESS_KEY
      secret_key: YOUR_RH_DATA_FOUNDATION_SECRET_KEY
```

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
      insecure: true
      access_key: YOUR_ROKS_ACCESS_KEY
      secret_key: YOUR_ROKS_SECRET_KEY
```

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.

5. You can retrieve the S3 access key and secret key for your cloud providers with the following commands:

```
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}')

echo $ACCESS_KEY

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}')

echo $SECRET_KEY
```

You must decode, edit, and encode your **base64** string in the secret.

1.2.2.1. Creating the MultiClusterObservability CR

Complete the following steps to create the **MultiClusterObservability** custom resource (CR) for your managed 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, 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. Click the **Grafana link** that is near the console header, from either the console *Overview* page or the *Clusters* page.
   Note: 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 functionalities 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 with Red Hat OpenShift Container Platform version 4.8 or later. After you install Red Hat Advanced Cluster Management with observability enabled, alerts from OpenShift Container Platform v4.8 and later are automatically forwarded to the hub cluster.

See Forwarding alerts to learn more.

1.2.3. 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**. Be sure to create an image pull-secret named, **multiclusterhub-operator-pull-secret** in the **open-cluster-management-observability** project.

Create your object storage secret named, **thanos-object-storage** in the **open-cluster-management-observability** project. Enter the object storage secret details, then click **Create**. **Note**: See step 4 of the Enabling observability section to view an example of a secret.

Create the **MultiClusterObservability** CR instance. When you receive the following message, the obseravbility service is enabled successfully from OpenShift Container Platform: **Observability components are deployed and running**.

1.2.3.1. Using the external metric query

Observability provides an external API for metrics to be queried through the OpenShift route, **rbac-query-proxy**. View the following tasks to use **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
  ```

- To access the **rbac-query-proxy** route, you must have an 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.

- 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
  ```

- 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.

- You can also replace certificates for the **rbac-query-proxy** route:

  - See OpenSSL commands for generating a certificate to create certificates. When you customize the **csr.cnf**, update the **DNS.1** to the hostname for the **rbac-query-proxy** route.

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

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

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

1.2.4. Disabling observability

To disable the observability service, uninstall the **observability** resource. From the OpenShift Container Platform console navigation, select **Operators** > **Installed Operators** > **Advanced Cluster Manager for Kubernetes**. Remove the **MultiClusterObservability** custom resource.

To learn more about customizing the observability service, see Customizing observability.

1.3. CUSTOMIZING OBSERVABILITY

Review the following sections to learn more about customizing, managing, and viewing data that is collected by the observability service.

Collect logs about new information that is created for observability resources with the **must-gather** command. For more information, see the *Must-gather* section in the Troubleshooting documentation.

- Creating custom rules
- Configuring AlertManager
- Adding custom metrics
- Removing default metrics
- Adding *advanced* configuration
- Updating the *multiclusterobservability* CR replicas from the console
- Forwarding alerts
- Customizing route certification
- Viewing and exploring data
  - Viewing the etcd table
  - Viewing the cluster fleet service-level overview for the Kubernetes API server dashboard
  - Viewing the cluster service-level overview for the Kubernetes API server dashboard
- Disable observability

1.3.1. Creating custom rules

Create custom rules for the observability installation by adding Prometheus recording rules and alerting rules to the observability resource. For more information, see Prometheus configuration.

- Recording rules provide you the ability to precalculate, or computate expensive expressions as needed. The results are saved as a new set of time series.
- Alerting rules provide you the ability to specify the alert conditions based on how an alert should be sent to an external service.

Define custom rules with Prometheus to create alert conditions, and send notifications to an external messaging service. **Note**: When you update your custom rules, **observability-thanos-rule** pods are restarted automatically.

Create a ConfigMap named **thanos-ruler-custom-rules** in the **open-cluster-management-observability** namespace. The key must be named, **custom_rules.yaml**, as shown in the following example. You can create multiple rules in the configuration.

- By default, the out-of-the-box alert rules are defined in the **thanos-ruler-default-rules** ConfigMap in the **open-cluster-management-observability** namespace.
  For example, you can create a custom alert rule that notifies you when your CPU usage passes your defined value. Your YAML might resemble the following content:

```
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
```

- You can also create a custom recording rule within the **thanos-ruler-custom-rules** ConfigMap.
  For example, you can create a recording rule that provides you the ability to get the sum of the container memory cache of a pod. Your YAML might resemble the following content:

```
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: If this is the first new custom rule, it is created immediately. For changes to the ConfigMap, the configuration is automatically reloaded. The configuration is reloaded due to the **config-reload** within the **observability-thanos-ruler** sidecar.

To verify that the alert rules are functioning appropriately, launch the Grafana dashboard, navigate to the **Explore** page, and query **ALERTS**. The alert is only available in Grafana if the alert is initiated.

1.3.2. 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.

1.3.3. Adding custom metrics

Add metrics to the **metrics_list.yaml** file, to be collected from managed clusters.

Before you add a custom metric, verify that **mco observability** is enabled with the following command: **oc get mco observability -o yaml**. Check for the following message in the **status.conditions.message** reads: **Observability components are deployed and running**.

Create a file named **observability-metrics-custom-allowlist.yaml** and add the name of the custom metric to the **metrics_list.yaml** parameter. Your YAML for the ConfigMap might resemble the following content:

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: observability-metrics-custom-allowlist
data:
  metrics_list.yaml: |
    names:
      - node_memory_MemTotal_bytes
    rules:
    - 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))
```

- In the **names** section, add the name of the custom metrics that is to be collected from the managed cluster.
- In the **rules** section, 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.
- The **names** and **rules** sections are optional. You can use either one or both of the sections.

Create the **observability-metrics-custom-allowlist** ConfigMap in the **open-cluster-management-observability** namespace with the following command: **oc apply -n open-cluster-management-observability -f observability-metrics-custom-allowlist.yaml**.

Verify that data from your custom metric is being collected by querying the metric from the **Explore** page, from the Grafana dashboard. You can also use the custom metrics in your own dashboard. For more information about viewing the dashboard, see Designing your Grafana dashboard.

1.3.4. Removing default metrics

If you want data to not be collected in your managed cluster for a specific metric, remove the metric from the **observability-metrics-custom-allowlist.yaml** file. When you remove a metric, the metric data is not collected in your managed clusters. As mentioned previously, first verify that **mco observability** is enabled.

Add the name of the default metric to the **metrics_list.yaml** parameter with a hyphen **-** at the start of the metric name. For example, **-cluster_infrastructure_provider**.

Create the **observability-metrics-custom-allowlist** ConfigMap in the **open-cluster-management-observability** namespace with the following command: **oc apply -n open-cluster-management-observability -f observability-metrics-custom-allowlist.yaml**.

Verify that the specific metric is not being collected from your managed clusters. When you query the metric from the Grafana dashboard, the metric is not displayed.

1.3.5. Adding *advanced* configuration

Add the **advanced** configuration section to update the retention for each observability component, according to your needs.

Edit the **MultiClusterObservability** CR and add the **advanced** section with the following command: **oc edit mco observability -o yaml**. Your YAML file might resemble the following contents:

```
spec:
  advanced:
    retentionConfig:
      blockDuration: 2h
      deleteDelay: 48h
      retentionInLocal: 24h
      retentionResolutionRaw: 30d
      retentionResolution5m: 180d
      retentionResolution1h: 0d
    receive:
      resources:
        limits:
          memory: 4096Gi
      replicas: 3
```

For descriptions of all the parameters that can added into the **advanced** configuration, see the Observability API.

1.3.6. Updating the *multiclusterobservability* CR 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 (CR), 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** CR, see the Observability API.

1.3.7. 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://****
```

To learn more, see the Prometheus Alertmanager documentation.

1.3.8. Customizing route certification

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>**.

1.3.9. Viewing and exploring data

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 *cluster_infrastructure_provider* from a single node 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 Management Workload Partitioning for more information.

1.3.9.1. Viewing the etcd table

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.

1.3.9.2. Viewing the cluster fleet service-level overview for the Kubernetes API server dashboard

View the cluster fleet Kubernetes API service-level overview from the hub cluster dashboard in Grafana.

After you navigate to the Grafana dashboard, access the managed dashboard menu by selecting **Kubernetes** > **Service-Level Overview** > **API Server**. The *Fleet Overview* and *Top Cluster* details are displayed.

View 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.

1.3.9.3. Viewing the cluster service-level overview for the Kubernetes API server dashboard

View the Kubernetes API service-level overview table from the hub cluster dashboard in Grafana.

After you navigate to the Grafana dashboard, access the managed dashboard menu by selecting **Kubernetes** > **Service-Level Overview** > **API Server**. The *Fleet Overview* and *Top Cluster* details are displayed.

View the error budget for the past seven or 30-day period, the remaining downtime, and trend.

1.3.10. Disable observability

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

1.3.10.1. Disable 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
```

1.3.10.2. Disable observability on a single cluster

Disable observability by removing observability components on specific managed clusters. Add the **observability: disabled** label to the **managedclusters.cluster.open-cluster-management.io** custom resource.

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.

For more information on monitoring data from the console with the observability service, see Observing environments introduction.

1.4. DESIGNING YOUR GRAFANA DASHBOARD

You can design your Grafana dashboard by creating a **grafana-dev** instance.

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

1.4.1. Setting up the Grafana developer instance

First, clone the **stolostron/multicluster-observability-operator/** repository, so that you are able to run the scripts that are in the **tools** folder. Complete the following steps to set up the Grafana developer instance:

1. 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
ingress.extensions/grafana-dev created
persistentvolumeclaim/grafana-dev created
```

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

   a. Select the Grafana URL, **https://$ACM_URL/grafana-dev/** 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.

1.4.2. Design 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**.

1.4.2.1. Design your Grafana dashboard with a ConfigMap

Complete the following steps to design your Grafana dashboard with a ConfigMap:

1. 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:

   a. Select a dashboard and click the **Dashboard settings** icon.

   b. Click the **JSON Model** icon from the navigation panel.

   c. Copy the dashboard JSON data and paste it in the ***data*** section.

   d. 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
```

Note: 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.

1.4.3. 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
service "grafana-dev" deleted
ingress.extensions "grafana-dev" deleted
persistentvolumeclaim "grafana-dev" deleted
```

1.5. 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 OpenShift Cluster Manager. 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

1.5.1. 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 OpenShift Cluster Manager, must be able to manage all the Red Hat Advanced Cluster Management managed clusters in OpenShift Cluster Manager.

1.5.2. Red Hat Insights from the Red Hat Advanced Cluster Management console

Continue reading to view functionality descriptions of the integration:

- When you select a cluster from the *Clusters* page, you can select the **Number of identified issues** from the *Status* card. The *Status* card displays information about *Nodes*, *Applications*, *Policy violations*, and *Identified issues*. 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*.

- After you click the number, the *Potential issue* side panel is displayed. A summary and chart of the total issues are displayed in the 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*.

- From the *Description* section, you can select the link to the vulnerability. View steps to resolve your vulnerability by selecting the *How to remediate* tab. You can also view why the vulnerability occurred by clicking the *Reason* tab.

See Managing insight **PolicyReports** for more information.

1.6. 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.

View the following sections to learn how to manage and view insight **PolicyReports**:

- Searching for insight policy reports

- Viewing identified issues from the console

1.6.1. Searching for insight policy reports

You can search for a specific insight **PolicyReport** that has a violation, across your managed clusters.

After you log into your Red Hat Advanced Cluster Management hub cluster, click the *Search* icon in the console header to navigate to the *Search* page. Enter the following query: **kind:policyreport**.

Note: The **PolicyReport** name matches the name of the cluster.

You can also further specify your query by 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.

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
```

1.6.2. Viewing identified issues from the console

You can view the identified issues on a specific cluster.

After you log into your Red Hat Advanced Cluster Management cluster, select **Overview** from the navigation menu. Select a severity to view the **PolicyReports** that are associated with that severity. Details of the cluster issues and the severities are displayed from the *Cluster issues* summary card.

Alternatively, you can select **Clusters** from the navigation menu. Select a managed cluster from the table to view more details. From the *Status* card, view the number of identified issues.

Select the number of potential issues to view the severity chart and recommended remediations for the issues. Click the link to the vulnerability to view steps on *How to remediate* and the *Reason* for the vulnerability.

Note: After the issue is resolved, the Red Hat Insights are received by Red Hat Advanced Cluster Management every 30 minutes and Red Hat Insights is updated every two hours.

Be sure to verify which component sent the alert message from the **PolicyReport**. Navigate to the *Governance* page and select a specific **policyreport**. Select the *Status* tab and click the **View details** link to view the **PolicyReport** YAML file.

Locate the **source** parameter, which informs you of the component that sent the violation. The value options are **grc** and **insights**.

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