# Red Hat Advanced Cluster Management for Kubernetes 2.2

## Observing environments

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

Last Updated: 2022-05-18

# Red Hat Advanced Cluster Management for Kubernetes 2.2 Observing environments

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

## Legal Notice

## Abstract

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

# Table of Contents

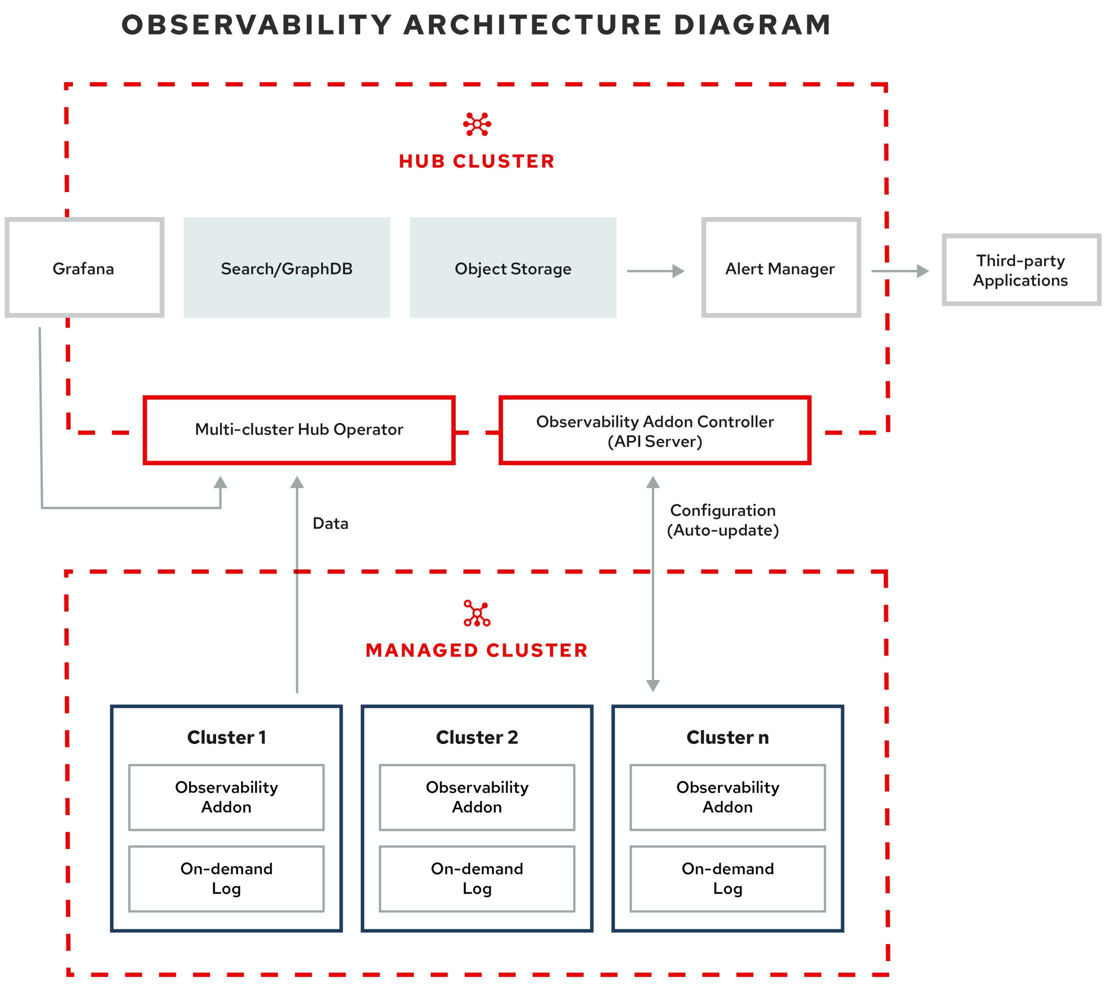
# CHAPTER 1. OBSERVING ENVIRONMENTS INTRODUCTION

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

## 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**, 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
- Observability certificates
- Metric types
- Observability pod capacity requests
- Persistent stores used in the observability service

### 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 stable object stores:

- Amazon S3 (or other S3 compatible object stores like Ceph)
- Google Cloud Storage
- Azure storage
- Red Hat OpenShift Container Storage

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.

When observability is enabled in a hub cluster, metrics are collected by handling the hub cluster as a managed cluster called **local-cluster**.

**Note**: In Red Hat Advanced Cluster Management the **metrics-collector** is only supported for Red Hat OpenShift Container Platform 4.x clusters.

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 6.4.x 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.

#### 1.1.1.1. Observability certificates

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

- Components on your hub cluster automatically restart to retrieve the renewed certificate.
- Red Hat Advanced Cluster Management sends the renewed certificates to managed clusters.

- The **metrics-collector** restarts to mount the renewed certificates.
  **Note: metrics-collector** can push metrics to the hub cluster before and after certificates are removed. For more information about refreshing certificates across your clusters, see Refresh a managed certificate.

### 1.1.1.2. Metric types

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

- The **visual_web_terminal_sessions_total** is collected on the hub cluster.
- The **acm_managed_cluster_info** is collected on each managed cluster and sent to the hub cluster.

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

### 1.1.1.3. Observability pod capacity requests

Observability components require 2636mCPU and 11388Mi memory to install the observability service. View the following table of the pod capacity requests that is 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 |
|---|---|---|---|---|---|---|
| alertmanager | alertmanager | 4 | 200 | 3 | 12 | 600 |
| | config-reloader | 4 | 25 | 3 | 12 | 75 |
| grafana | grafana | 4 | 100 | 2 | 8 | 200 |
| | grafana-dashboard-loader | 4 | 50 | 2 | 8 | 100 |
| observability-observatorium-observatorium-api | observatorium-api | 20 | 128 | 2 | 40 | 256 |

| Deployment or StatefulSet | Container name | CPU (mCPU) | Memory (Mi) | Replicas | Pod total CPU | Pod total memory |
|---|---|---|---|---|---|---|
| observability-observatorium-thanos-compact | thanos-compact | 100 | 512 | 1 | 100 | 512 |
| observability-observatorium-thanos-query | thanos-query | 300 | 1024 | 2 | 600 | 2048 |
| observability-observatorium-thanos-query-frontend | thanos-query-frontend | 100 | 256 | 2 | 200 | 512 |
| observability-observatorium-thanos-receive-controller | thanos-receive-controller | 4 | 32 | 1 | 4 | 32 |
| observability-observatorium-thanos-receive-default | thanos-receive | 300 | 512 | 3 | 900 | 1536 |
| observability-observatorium-thanos-rule | thanos-rule | 50 | 512 | 3 | 150 | 1536 |
| | configmap-reloader | 4 | 25 | 3 | 12 | 75 |
| observability-observatorium-thanos-store-memcached | memcached | 45 | 128 | 3 | 135 | 384 |
| | exporter | 5 | 50 | 3 | 15 | 150 |

| Deployment or StatefulSet | Container name | CPU (mCPU) | Memory (Mi) | Replicas | Pod total CPU | Pod total memory |
|---|---|---|---|---|---|---|
| observability-observatorium-thanos-store-shard | thanos-store | 100 | 1024 | 3 | 300 | 3072 |
| observatorium-operator | observatorium-operator | 100 | 100 | 1 | 100 | 100 |
| rbac-query-proxy | rbac-query-proxy | 20 | 100 | 2 | 40 | 200 |

### 1.1.2. Persistent stores used in the observability service

When you install Red Hat Advanced Cluster Management the following persistent volumes are created:

**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 notificationn 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. |

| | |
|---|---|
| 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. |
| 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 .

## 1.2. 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
- 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 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 Container Storage
  **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. 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
   ```

   a. 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.

- For OpenShift Container Storage, 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_OCS_BUCKET
      endpoint: YOUR_OCS_ENDPOINT
      insecure: true
      access_key: YOUR_OCS_ACCESS_KEY
      secret_key: YOUR_OCS_SECRET_KEY
```

For more details, see Installing OpenShift Container Storage .

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

```
ACCESS_KEY=$(oc -n <your-object-storage> get secret <object-storage-secret> -o yaml
| grep AccessKey | awk '{print $2}' | base64 --decode)

echo $ACCESS_KEY

SECRET_KEY=$(oc -n <your-object-storage> get secret <object-storage-secret> -o yaml
| grep SecretKey | awk '{print $2}' | base64 --decode)

echo $SECRET_KEY
```

#### 1.2.2.1. Creating the *MultiClusterObservability* CR

Complete the following steps to create the **MultiClusterObservability** custom resource (CR):

1. Create the **MultiClusterObservability** custom resource (mco CR) for your managed cluster by completing the following steps:

   a. 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/v1beta1
kind: MultiClusterObservability
metadata:
  name: observability #Your customized name of MulticlusterObservability CR
spec:
  availabilityConfig: High # Available values are High or Basic
  enableDownSampling: false # The default value is false. This is not recommended as
querying long-time ranges without non-downsampled data is not efficient and useful.
  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: true # EnableMetrics indicates the observability addon push metrics to
hub server
    interval: 30 # Interval for the observability addon push metrics to hub server
  retentionResolution1h: 30d # How long to retain samples of 1 hour in bucket
  retentionResolution5m: 14d
  retentionResolutionRaw: 5d
  storageConfigObject: # Specifies the storage to be used by Observability
    metricObjectStorage:
      name: thanos-object-storage
      key: thanos.yaml
    statefulSetSize: 10Gi # The amount of storage applied to the Observability
StatefulSets, i.e. Amazon S3 store, Rule, compact and receiver.
    statefulSetStorageClass: gp2
```

You might want to modify the value for the **retentionResolution** parameter. By default, downsampling is disabled. For more information, see Thanos Downsampling resolution and retention. Depending on the number of managed clusters, you might want to update **statefulSetSize**, see Observability API for more information.

b. 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.

c. 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.

2. To validate that the observability service is enabled, launch the Grafana dashboards to make sure the data is populated. Complete the following steps:

   a. Log in to the Red Hat Advanced Cluster Management console.

   b. From the navigation menu, select **Observe environments** > **Overview**.

c. Click the Grafana link that is near the console header to view the metrics from your managed clusters.
**Note**: If you want to exclude specific managed clusters from collecting the observability data, add the following cluster label to your clusters: **observability: disabled**.

### 1.2.3. Disabling observability

To disable the observability service, uninstall the **observability** resource. See step 1 of Removing a MultiClusterHub instance by using commands for the procedure.

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 rules for AlertManager
- Adding custom metrics
- Viewing and exploring data
- Disable observability

### 1.3.1. Creating custom rules

You can 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-observatorium-thanos-rule** pods are restarted automatically.

Complete the following steps to create a custom rule:

1. Log in to your Red Hat Advanced Cluster Management hub cluster.
2. 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:

```
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, you must restart the observability pods with the following command: **kubectl rollout restart statefulset observability-observatorium-thanos-rule -n open-cluster-management-observability**.

3. If you want to verify that the alert rules is functioning appropriately, complete the following steps:

   a. Access your Grafana dashboard and select the **Explore** icon.

   b. In the Metrics exploration bar, type in "ALERTS" and run the query. All the ALERTS that are currently in pending or firing state in the system are displayed.

   c. If your alert is not displayed, revisit the rule to see if the expression is accurate.

A custom rule is created.

### 1.3.2. Configuring rules for 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.

Complete the following steps to add custom metrics:

1. Log in to your cluster.

2. Verify that **mco observability** is enabled. Check for the following message in the **status.conditions.message** reads: **Observability components are deployed and running**. Run the following command:

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

3. Create a new file **observability-metrics-custom-allowlist.yaml** with the following content. Add the name of the custom metric to the **metrics_list.yaml** parameter. For example, add

**node_memory_MemTotal_bytes** to the metric list. 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
```

4. Create the **observability-metrics-custom-allowlist** ConfigMap in the **open-cluster-management-observability** namespace by running the following command:

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

5. Verify that your custom metric is being collected from your managed clusters by viewing the metric on the Grafana dashboard. From your hub cluster, select the **Grafana dashboard** link.

6. From the Grafana search bar, enter the metric that you want to view.

Data from your custom metric is collected.

### 1.3.4. Viewing and exploring data

View the data from your managed clusters by accessing Grafana. Complete the following steps to view the Grafana dashboards from the console:

1. Log in to your Red Hat Advanced Cluster Management hub cluster.

2. From the navigation menu, select **Observe environments** > **Overview** > **Grafana link**.
   You can also access Grafana dashboards from the *Clusters* page. From the navigation menu, select **Automate infrastructure** > **Clusters** > **Grafana**.

3. Access the Prometheus metric explorer by selecting the **Explore** icon from the Grafana navigation menu.

### 1.3.5. Disable observability

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

#### 1.3.5.1. 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:
  availabilityConfig: High # Available values are High or Basic
  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
```

### 1.3.5.2. Disable observability on a single cluster

Disable observability on specific managed clusters by completing one of the following procedures:

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

- From the Red Hat Advanced Cluster Management console *Clusters* page, add the **observability: disabled** label by completing the following steps:

  1. In the Red Hat Advanced Cluster Management console navigation, select **Automate infrastructure** > **Clusters**.

  2. Select the name of the cluster for which you want to disable data collection that is sent to observability.

  3. Select **Labels**.

  4. Create the label that disables the observability collection by adding the following label:

     ```
     observability=disabled
     ```

  5. Select **Add** to add the label.

  6. Select **Done** to close the list of labels.

**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.

### 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 a Grafana administrator. For example, after you log in using **kubeadmin**, run the 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 ***metadata*** section.

   d. Modify the ***name*** and replace ***$your-dashboard-name***. 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
```