IBM Netcool Operations Insight Event Integrations Operator 3.10.0

Reference Guide
December 6, 2023

SC28-3144-17

Note

Before using this information and the product it supports, read the information in Appendix A, “Notices and Trademarks,” on page 155.

Edition notice

This edition (SC28-3144-17) applies to version 3.10.0 of IBM® Netcool® Operations Insight® Event Integrations Operator and to all subsequent releases and modifications until otherwise indicated in new editions.

This edition replaces SC28-3144-16.

Contents

About this guide...... v
Document control page...... v

Chapter 1. IBM Netcool Operations Insight Event Integrations Operator......1
What's new...... 1
Prerequisites......15
Resources required......15
Ports information and firewall requirements......16
Security context constraints......16
Gathering information for IBM Support...... 17
Role-based access control...... 18
Operator scope...... 20
Storage...... 20
Installing the IBM Netcool Operations Insight Event Integrations Operator...... 21
Installing the IBM Netcool Operations Insight Event Integrations Operator to a private container registry...... 26
Verifying the installation......38
Operator Upgrade and Rollback process...... 39
Upgrading with the OLM UI......39
Upgrading with the CLI......40
Rolling back process...... 41
Installing integration resources using the CLI......42
Configuring the global operands resources ConfigMap......42
Providing custom probe rules in a ConfigMap......44
Viewing the health of an operator...... 45
Enabling probe self-monitoring......46
Monitoring event statistics...... 47
Pulling images from IBM Entitled Registry...... 48
Listing images in IBM Entitled Registry......48
Configuring custom resources......49
Updating an integration instance...... 49
Deleting an integration instance......49
Uninstalling the operator......49
Using the OLM UI......49
Using the CLI with CASE......50
Using the CLI (native)...... 50
Limitations......51
Troubleshooting...... 52
Known issues...... 55

Chapter 2. Deploying integrations...... 61
Connecting with IBM Cloud Pak for AIOps...... 61
Connecting on-premise probes and gateways to IBM Cloud Pak for AIOps...... 61
Connecting probes and gateways on Red Hat OpenShift......62
Deploying the Gateway for JDBC Integration......65
Deploying the Probe for Prometheus Integration...... 75
Deploying the Probe for Kafka Integration...... 80
Deploying the Gateway for Kafka......100
Deploying the Probe for Pulsar Integration......107
Deploying the Gateway for Pulsar Integration...... 115

Deploying the Generic Webhook Probe.......119
Common pre-installation tasks for the IBM SevOne, IBM Turbonomic, and Datadog probes.......139
Deploying the Probe for IBM SevOne NPM Integration.......140
Deploying the Probe for Turbonomic Integration.......144
Deploying the Probe for Datadog Integration.......149

Appendix A. Notices and Trademarks.......155
Notices.......155
Trademarks.......156

About this guide

The following sections contain important information about using this guide.

Document control page

Use this information to track changes between versions of this guide.

The IBM Netcool Operations Insight Event Integrations Operator documentation is provided in softcopy format only. To obtain the most recent version, visit the IBM Tivoli® Knowledge Center:

https://www.ibm.com/support/knowledgecenter/SSSHTQ_int/omnibus/operators/all_operators/ops_intro.html

Table 1. Document modification history

Document version	**Publication date**	**Comments**
SC28-3144-00	June 26, 2020	First IBM publication.
SC28-3144-01	October 30, 2020	Updated for version 1.1.0. The following topics updated: • "Installing the IBM Netcool Operations Insight Event Integrations Operator" on page 21 • "Using the CLI with CASE" on page 50 • "Using the CLI (native)" on page 50 The following topics added: • "Installing the IBM Netcool Operations Insight Event Integrations Operator to a private container registry" on page 26 • "Operator Upgrade and Rollback process" on page 39 • "Rolling back process" on page 41 • "Upgrading with the CLI" on page 40 • "Upgrading with the OLM UI" on page 39 • "Installing integration resources using the CLI" on page 42
SC28-3144-02	December 11, 2020	Updated for version 1.2.0. The following topics updated: • "Troubleshooting" on page 52 The following topics added: • "Deploying the Probe for Kafka Integration" on page 80 • "Integrating with IBM Event Streams for IBM Cloud" on page 94 • "Customizing probe rules in ConfigMap" on page 99

Table 1. Document modification history (continued)

Document version	**Publication date**	**Comments**
`SC28-3144-03`	March 31, 2021	Updated for version 2.0.0. The following topics updated: • “What's new” on page 1 • “Troubleshooting” on page 52 The following topics added: • “Connecting with IBM Cloud Pak for AIOps” on page 61 • “Integrating with IBM Event Streams for Red Hat Openshift” on page 88
`SC28-3144-04`	December 10, 2021	Updated for version 3.2.0. The following topics updated: • “What's new” on page 1 The following topics added: • “Ports information and firewall requirements” on page 16 • “Verifying the installation” on page 38 • “Deploying the Gateway for JDBC Integration” on page 65 • “Deploying the Probe for Pulsar Integration” on page 107 • “Deploying the Gateway for Pulsar Integration” on page 115
`SC28-3144-05`	January 14, 2022	Updated for version 3.2.3. The following topics updated: • “What's new” on page 1
`SC28-3144-06`	March 30, 2022	Updated for version 3.3.0. The following topics updated: • “What's new” on page 1
`SC28-3144-07`	June 29, 2022	Updated for version 3.4.0. The following topic updated: • “What's new” on page 1 The following topic created: • “Deploying the Generic Webhook Probe” on page 119
`SC28-3144-08`	July 29, 2022	Updated for version 3.4.1. The following topic updated: • “What's new” on page 1

Table 1. Document modification history (continued)

Document version	**Publication date**	**Comments**
SC28-3144-09	September 28, 2022.	Updated for version 3.5.0. Support for Probe for IBM Turbonomic® added. CEM Probe and CEM Gateway deprecated on September 30, 2022. The following topic updated: • "What's new" on page 1 The following topics created: • "Deploying the Probe for Turbonomic Integration" on page 144
SC28-3144-10	December 14, 2022.	Updated for version 3.6.0. Support for Gateway for Kafka added. The following topic updated: • "What's new" on page 1 • "Configuring the global operands resources ConfigMap" on page 42 • "Limitations" on page 51 The following topics created: • "Deploying the Gateway for Kafka" on page 100
SC28-3144-11	January 25, 2023.	Updated for version 3.6.1. The following topic updated: • "What's new" on page 1
SC28-3144-12	March 29, 2023.	Updated for version 3.7.0. CEM Probe and CEM Gateway deprecated. The following topics updated: • "What's new" on page 1 • "Installing the IBM Netcool Operations Insight Event Integrations Operator" on page 21 • "Installing the IBM Netcool Operations Insight Event Integrations Operator to a private container registry" on page 26 The following topics created: • "Providing custom probe rules in a ConfigMap" on page 44 • "Enabling probe self-monitoring" on page 46 • "Monitoring event statistics" on page 47
SC28-3144-13	April 28, 2023.	Updated for version 3.7.1. The following topic updated: • "What's new" on page 1

Table 1. Document modification history (continued)

Document version	**Publication date**	**Comments**
SC28-3144-14	June 28, 2023.	Updated for version 3.8.0. The following topic updated: "What's new" on page 1. Descriptions for the new **`spec.helmValues.probe.recordData`** and **`spec.helmValues.probe.configPVC.dataBackupFile`** properties added to "Deploying the Generic Webhook Probe" on page 130. "Deploying the Probe for Datadog Integration" on page 149 added.
SC28-3144-15	July 26, 2023.	Updated for version 3.8.1. The following topic updated: "What's new" on page 1.
SC28-3144-16	October 25, 2023.	Updated for version 3.9.0. The following topic updated: "What's new" on page 1.
SC28-3144-17	December 6, 2023.	Updated for version 3.10.0. The following topic updated: "What's new" on page 1.
SC28-3144-18	May 16, 2025.	Updated for version 3.11.0. The following topic updated: "What's new" on page 1.

Chapter 1. IBM Netcool Operations Insight Event Integrations Operator

The IBM Netcool Operations Insight Event Integrations Operator can deploy multiple integrations, for example: it can run instances of the Probe for Prometheus, Probe for Kafka, Probe for IBM SevOne®, and Probe for IBM Turbonomic.

The IBM Netcool Operations Insight Event Integrations Operator currently supports the following integrations:

- IBM Netcool/OMNIbus Probe for Prometheus

  Creates the webhook endpoints to receive notification in the form of HTTP POST requests from Prometheus Alert Manager.
- IBM Netcool/OMNIbus Probe for Kafka

  Creates a Kafka consumer to subscribe messages from a Kafka topic.
- IBM Netcool/OMNIbus Probe for Pulsar Integration

  Creates a Pulsar client, subscribes and processes events from a Pulsar broker.
- IBM Netcool/OMNIbus Gateway for Pulsar Integration

  Creates a Pulsar producer that processes events and alerts from IBM Netcool/OMNIbus ObjectServer and forwards them to Pulsar.
- IBM Netcool/OMNIbus JDBC Gateway Integration

  Processes events and alerts from IBM Netcool/OMNIbus ObjectServer and forwards them to a target Relational Database Management System (RDBMS) database.
- IBM Netcool/OMNIbus Probe for Generic Webhook Integration

  IBM Netcool/OMNIbus Probe for Generic Webhook creates webhook endpoint to receive notification in the form of HTTP POST requests with JSON payloads from event source.

Each of the NOI integrations is defined as a separate resource kind.

What's new

This section describes what's new in IBM Netcool Operations Insight Event Integrations CASE.

Version 3.11.0

Release date: April 16, 2025

Part number:???????

This CASE version contains IBM Netcool Operations Insight Event Integrations Operator version 3.13.0 compatible with IBM Netcool Operations Insight 1.6.14.

1. This operator version is compatible with Red Hat® OpenShift 4.14, 4.15, 4.16 and 4.17.

   Introduction of a new subscription channel version "v3.13" in the operator bundle. This allows users to select the operator version to install from the available subscription channels. If you have an older operator version installed, you can select the "v3.13" subscription channel to upgrade your operator to version 3.13.
2. Several vulnerabilities fixes included in this operator and operand version.
3. CVE-2024-28098, CVE-2024-29834, CVE-2024-27309, CVE-2024-29025, CVE-2024-47561

Version 3.10.0

Release date: July 31, 2024

Part number: M0JYGML

This CASE version contains IBM Netcool Operations Insight Event Integrations Operator version 3.12.0 compatible with IBM Netcool Operations Insight 1.6.13.

1. This operator version is compatible with Red Hat® OpenShift 4.14, 4.15 and 4.16.
2. Introduction of a new subscription channel version "v3.12" in the operator bundle. This allows users to select the operator version to install from the available subscription channels. If you have an older operator version installed, you can select the "v3.12" subscription channel to upgrade your operator to version 3.12.
3. Several vulnerabilities fixes included in this operator and operand version.

   CVE-2023-44487, CVE-2024-24786, CVE-2024-27309, CVE-2024-28098, CVE-2024-29025, CVE-2024-29834

Version 3.9.0

Release date: March 27, 2024

Part number: M0HK4ML

This CASE version contains IBM Netcool Operations Insight Event Integrations Operator version 3.11.0 compatible with IBM Netcool Operations Insight 1.6.12.

1. This operator version is compatible with Red Hat® OpenShift 4.12, 4.14 and 4.15.
2. Introduction of a new subscription channel version "v3.11" in the operator bundle. This allows users to select the operator version to install from the available subscription channels. If you have an older operator version installed, you can select the "v3.11" subscription channel to upgrade your operator to version 3.11.
3. Several vulnerabilities fixes included in this operator and operand version.

   CVE-2023-3978, CVE-2023-39325, CVE-2023-44487, CVE-2023-48795

Version 3.8.0

Release date: December 6, 2023

Part number: M0GG4ML

This CASE version contains IBM Netcool Operations Insight Event Integrations Operator version 3.10.0 compatible with IBM Netcool Operations Insight 1.6.11.

1. This operator version is compatible with Red Hat® OpenShift 4.12 and 4.14.
2. Introduction of a new subscription channel version `v3.10` in the operator bundle. This allows users to select the operator version to install from the available subscription channels. If you have an older operator version installed, you can select the `v3.10` subscription channel to upgrade your operator to version 3.10.
3. Introduced a new configurable **`enableHostnameVerification`** (default is `true`) HTTP request parameter to allow users to enable or disable the Webhook probe to perform hostname verification of target event source.
4. Several vulnerabilities fixes included in this operator and operand version:

   CVE-2023-4586, CVE-2023-44981, CVE-2023-43642

Version 3.7.0

Release date: October 25, 2023

Part number: M0G02ML

This CASE version contains IBM Netcool Operations Insight Event Integrations Operator version 3.9.0 compatible with IBM Netcool Operations Insight 1.6.10.

1. Several vulnerabilities fixes included in this operator and operand version:

   CVE-2023-2976, CVE-2023-33201, CVE-2023-34454, CVE-2023-34453, CVE-2023-34455, CVE-2023-34462, CVE-2023-35116, Several CVEs from Red Hat UBI 8 base image.

Version 3.6.1

Release date: July 26, 2023

Part number: N/A

Refer to the following Release Notice for a link to the CASE 3.6.1 release bundle: CASE 3.6.0 Release Notice

This CASE version contains IBM Netcool Operations Insight Event Integrations Operator version 3.8.1 compatible with IBM Netcool Operations Insight 1.6.9.

1. Fixed an issue where the Webhook Probe failed to perform resynchronization with Datadog when pagination was disabled.
2. Fixed an issue in the Webhook Probe to redact additional sensitive information from HTTP headers in the debug logs. This fix also applies to Prometheus Probe.
3. Added operators.openshift.io/infrastructure-features: '["disconnected"]' annotation into the Operator ClusterServiceVersion to indicate that the Operator supports being mirrored into a disconnected environment which does not have internet access.
4. Several vulnerabilities fixes included in this operator and operand version:

   CVE-2022-1471, CVE-2023-2976, CVE-2023-33201, CVE-2023-34453, CVE-2023-34454, CVE-2023-34455, CVE-2023-34462

Version 3.6.0

Release date: June 28, 2023

Part number: M0CW5ML

This CASE version contains IBM Netcool Operations Insight Event Integrations Operator version 3.8.0 compatible with IBM Netcool Operations Insight 1.6.9.

1. This operator version is compatible with Red Hat® OpenShift 4.10, 4.11 and 4.12.
2. Introduction of a new subscription channel version v3.8 in the operator bundle. This allows users to select the operator version to install from the available subscription channels. If you have an older operator version installed, you can select the v3.8 subscription channel to upgrade your operator to version 3.8.
3. `WebhookProbe` operand default version is increased to version 3.3.0. This version contains the following changes:
   a. Version 3.0.0 is longer supported. The operand will be upgraded to the latest version.
   b. Introduction of a new integration with Datadog using HTTP client. When the integration parameter is set to `datadog`, the probe rules file `configmap` will be created with preconfigured rules for the Datadog integration.
4. New configurable parameters introduced in the `WebhookProbe` API to configure the Webhook Probe:
   a. The **`probe.recordData`** (default is `nil`) allows users to specify a comma-separated list of attributes from the event to be recorded in the probe data backup file. The data recorded with the prefix `RecordData.` can be used by the probe to resolve transport properties using token with prefix and suffix of ++.

   b. The **`probe.configPVC.dataBackupFile`** (default is `nil`) allows users to specify the path to probe data backup file. If left unset, the probe stores the probe backup data in the default probe data backup file.
5. Several vulnerabilities fixes included in this operator and operand version:
   a. Several CVEs from Red Hat UBI 8 base image.

Version 3.5.1

Release date: April 28, 2023

Part number: N/A

Refer to the following Release Notice for a link to the CASE 3.5.1 release bundle: CASE 3.5.0 Release Notice

This CASE version contains IBM Netcool Operations Insight Event Integrations Operator version 3.7.1 compatible with IBM Netcool Operations Insight 1.6.8.

1. Upgraded HorizontalPodAutoscaler resource API version to `autoscaling/v2` from `autoscaling/v2beta1` of `PrometheusProbe`. This fixes the issue where the HorizontalPodAutoscaler resource is not created when running on Red Hat OpenShift 4.12.
2. Upgraded HorizontalPodAutoscaler resource API version to `autoscaling/v2` from `autoscaling/v1` of `KafkaProbe`, `PulsarProbe` and `WebhookProbe`. This fixes the issue where the HorizontalPodAutoscaler resource isn't created when running on Red Hat OpenShift 4.12.
3. Several vulnerabilities fixes included in this operator and operand version.
4. Several CVEs from Red Hat UBI 8 base image.

Version 3.5.0

Release date: March 29, 2023

Part number: M0BDHML

This CASE version contains IBM Netcool Operations Insight Event Integrations Operator version 3.7.0 compatible with IBM Netcool Operations Insight 1.6.8.

1. This operator version is compatible with Red Hat OpenShift 4.10, 4.11 and 4.12.
2. Introduction of a new subscription channel version **v3.7** in the operator bundle. This allows users to select the operator version to install from the available subscription channels. If you have an older operator version installed, you can select the **v3.7** subscription channel to upgrade your operator to version 3.7.
3. CEM Probe and CEM Gateway which were deprecated in version 3.5.0 is no longer supported in this operator version.
4. Cloud Pak Launch CLI script upgraded to v0.14.0 and Cloud Pak Airgap CLI script upgraded to v0.13.8 which brings several improvements and fixes.
5. Added Openshift `restricted-v2` Security Context Constraints (SCC) into the list of SCC in prerequisite check when using a CASE action script.
6. `KafkaProbe` operand default version is increased to version 3.6.0. With this version introduced, version 3.2.0 and 3.3.0 are no longer supported. The operand will be upgraded to the latest version.
7. `WebhookProbe` operand default version is increased to version 3.2.0.
8. `PulsarProbe` operand default version is increased to version 1.3.0. With this version introduced, version 1.0.0 is longer supported. The operand will be upgraded to the latest version.
9. Probe Self Monitoring and Event Statistics functionality is introduced in `KafkaProbe`, `PulsarProbe` and `WebhookProbe` resources. See “Enabling probe self-monitoring” on page 46 and “Monitoring event statistics” on page 47 for more info.
10. Several vulnerabilities fixes included in this operator and operand version:

11. Several CVEs from Red Hat UBI 8 base image.

Version 3.4.1

Release date: January 25, 2023

Part number: M09JWML

This CASE version contains IBM Netcool Operations Insight Event Integrations Operator version 3.6.1 compatible with IBM Netcool Operations Insight 1.6.7.

1. Several vulnerabilities fixes included in this operator and operand version:

   CVE-2022-41881, CVE-2022-41915, Several CVEs from Red Hat UBI 8 base image.

Version 3.4.0

Release date: December 14, 2022

Part number: M09JWML

This CASE version contains IBM Netcool Operations Insight Event Integrations Operator version 3.6.0 compatible with IBM Netcool Operations Insight 1.6.7.

1. Introduction of a new subscription channel version v3.6 in the operator bundle. This allows users to select the operator version to install from the available subscription channels. If you have an older operator version installed, you can select the v3.6 subscription channel to upgrade your operator to version 3.6.0.
2. Introduction of a new Gateway for Kafka Integration (KafkaGateway) that processes events and alerts from IBM Netcool/OMNIbus ObjectServer and forwards them to Kafka. You can deploy a Gateway for Kafka Integration by installing the KafkaGateway custom resource onto your cluster.
3. Added a new CloudPak metadata into the CASE specification file (case.yaml) indicating that all operand images are already listed as related images in the OLM bundles and catalog. You can now use oc adm catalog mirror command to mirror the images.
4. Fixed an issue where the operator deployment file (manager.yaml) was using an incorrect image repository. The image repositories has been updated with the public IBM container registry (cp.icr.io).
5. The operator controller now appends "ibm-entitlement-key" image pull secret into the managed service account resources to be consistent with other IBM Cloud® Paks. Users can create an image pull secret with the same name in the namespace to for the pods to use when pulling images from the IBM Container Registry (cp.icr.io). If a custom user service account is configured by setting the serviceAccountName in the custom resource specification, pods will use the sercive account resource as-is. Alternatively, you can specify a different pull secret name to append in the managed service account resources by setting the image.secretName parameter in the custom resource specification.
6. Fixed an issue in the Pulsar probe where the probe does not shut down gracefully causing the probe process to hang.
7. Webhook probe now responds with 204 No Content response code instead of 200 OK when respondWithContent is OFF. When respondWithContent is ON, the probe will respond with 200 OK response code to an incoming HTTP requests when there is no errors.
8. Several vulnerabilities fixes included in this operator and operand version:

   CVE-2022-3172, CVE-2022-25857, CVE-2022-38749, CVE-2022-38750, CVE-2022-38751, CVE-2022-38752, Several CVEs from Red Hat UBI 8 base image.

Version 3.3.1

Release date: October 26, 2022

Part number: M08VKML

This CASE version contains IBM Netcool Operations Insight Event Integrations Operator version 3.5.1 compatible with IBM Netcool Operations Insight 1.6.6.

1. The catalog index image updated to use the file based catalog instead of the legacy SQL database. This change is transparent to users as it only affects how the catalog registry is served for Operator Lifecycle Manager (OLM) to retrieve the operator bundle manifests.
2. The following vulnerabily fixes are included in this operator and operand version:

   Several CVEs from Red Hat UBI 8 base image.

Version 3.3.0

Release date: September 28, 2022

Part number: M0871ML

This CASE version contains IBM Netcool Operations Insight Event Integrations Operator version 3.5.0 compatible with IBM Netcool Operations Insight 1.6.6.

1. Support for IBM Turbonomic added.
2. CEM Probe and CEM Gateway deprecated

Version 3.2.1

Release date: July 29, 2022

Part number: M07G3ML

This CASE version contains IBM Netcool Operations Insight Event Integrations Operator version 3.4.1 compatible with IBM Netcool Operations Insight 1.6.5.1

1. Refreshed operator and operand images with latest RedHat UBI8.6 base image.
2. Fixed an issue when pinging ObjectServers running on FixPack 28 or older.
3. Fixed missing `probe.integration` parameter issue in `WebhookProbe` custom resource definition file when installed using the offline (air-gap) method.

Version 3.2.0

Release date: June 29, 2022

Part number: M06SLML

This CASE version contains IBM Netcool Operations Insight Event Integrations Operator version 3.4.0 compatible with IBM Netcool Operations Insight 1.6.5

1. This operator version supports Openshift 4.8, 4.9 and 4.10.
2. Introduction of a new subscription channel version `v3.4` in the operator bundle. This allows users to select the operator version to install from the available subscription channels. If you have an older operator version installed, you can select the `v3.4` subscription channel to upgrade your operator to version 3.4.0.
3. Introduction of a new Probe for Generic Webhook Integration (`WebhookProbe`) that creates a webhook endpoint to receive notifications in the form of HTTP POST requests with JSON payloads from event source. You can deploy a Probe for Generic Webhook Integration by installing the `WebhookProbe` custom resource onto your cluster.
4. Introduction of global operands resources `ConfigMap` to allow users to define pod resource limits and requests per operand kind in a single global `ConfigMap`. The resource settings from this `ConfigMap` are applied to all custom resource instance by find and override the resource settings set in the custom resource specification. Only one global operands resources `ConfigMap` is allowed per namespace. Deleting the `ConfigMap` will revert the pod resource settings to what is configured in each custom resource specification. For more details on how to configure the global operands resources `ConfigMap`, refer to "Configuring the global operands resources ConfigMap" on page 42.

5. Several vulnerabilities fixes included in this operator and operand version:

   CVE-2020-10663, CVE-2020-36518, CVE-2021-22573, CVE-2022-24823, PRISMA-2021-0213 and several CVE on base UBI 8 image.

Version 3.1.1

This CASE version contains IBM Netcool Operations Insight Event Integrations Operator version 3.3.1 compatible with IBM Netcool Operations Insight 1.6.4

1. Several vulnerabilities fixes included in this operator and operand version:
   a. Several CVE on base UBI 8 image.
   b. CVE-2020-36518

Version 3.1.0

Release date: March 30, 2022

Part number: M05H4ML

This CASE version contains IBM Netcool Operations Insight Event Integrations Operator version 3.3.0 compatible with IBM Netcool Operations Insight 1.6.4

1. This operator version supports Openshift 4.7, 4.8 and 4.9.
2. Introduction of a new subscription channel version "v3.3" in the operator bundle. This allows users to select the operator version to install from the available subscription channels. If you have an older operator version installed, you can select the "v3.3" subscription channel to upgrade your operator to version 3.3.0.
3. The operator source codes has been udpated to use the Operator SDK v1.2 project layout. There are several notable changes due to this upgrade in the operator and the CASE:
   a. Operator pod now runs `manager` process in the container.
   b. The operator pod logs formatting has changed slightly for better readability.
4. Several CASE files in the `netcoolIntegrationsOperatorSetup`, specifically the `ibm-netcool-integrations/inventory/netcoolIntegrationsOperatorSetup/files/op-cli/deploy` directory has been renamed following the new Operator SDK v1.2 generated files:
   a. Operator deployment resource file renamed to `manager.yaml` (formerly `operator.yaml`)
   b. CustomResourceDefinition files in `ibm-netcool-integrations/inventory/netcoolIntegrationsOperatorSetup/files/op-cli/deploy/crds` has been renamed to omit the `_crd` suffix in the filename.
      - gateways.integrations.noi.ibm.com_cemgateways.yaml
      - gateways.integrations.noi.ibm.com_jdbcgateways.yaml
      - gateways.integrations.noi.ibm.com_pulsargateways.yaml
      - probes.integrations.noi.ibm.com_cemprobes.yaml
      - probes.integrations.noi.ibm.com_kafkaprobes.yaml
      - probes.integrations.noi.ibm.com_prometheusprobes.yaml
      - probes.integrations.noi.ibm.com_pulsarprobes.yaml
   c. CustomResource files in `ibm-netcool-integrations/inventory/netcoolIntegrationsOperatorSetup/files/op-cli/deploy/crs` has been shortened (the API domain deleted). New files names are:
      - gateways_v1_jdbcgateway.yaml
      - gateways_v1_pulsargateway.yaml
      - gateways_v1beta1_cemgateway.yaml

    - probes_v1_kafkaprobe.yaml
    - probes_v1_pulsarprobe.yaml
    - probes_v1beta1_cemprobe.yaml
    - probes_v1beta1_prometheusprobe.yaml
5. The operator now supports All Namespaces install mode when installed using the Operator Lifecycle Manager. You can now install a single instance of the operator which watches all namespaces in the cluster. This is recommended for fresh installation because there should only be a single instance of the operator running in the cluster. If you have an existing installation, you can opt to continue with the single namespace install mode or you need to uninstall all existing operator and re-install with the All Namespace mode to avoid conflict.
6. Fixed an issue where users upgrading from version 2.0.0 or 2.1.0 doesn't see their operand images upgraded in the pods even when `useDefaultOperandImages` is set to `true`. Added several environment variables in the operator pod log to map the images for the operator to update the operand pods.
7. Cloud Pak Launch CLI script upgraded to v0.11.0 and Cloud Pak Airgap CLI script upgraded to v0.13.4 which brings several improvements and fixes.
8. Several vulnerabilities fixes included in this operator and operand version:

   CVE-2021-37136, CVE-2021-37137, CVE-2015-5237, PRISMA-2021-0213, CVE-2018-14040, CVE-2020-8559, CVE-2020-8559, CVE-2020-26160, CVE-2021-33194, CVE-2021-38561, CVE-2021-43565, CVE-2019-11358, CVE-2021-27918, CVE-2020-28852, WS-2021-0200, CVE-2020-10752, CVE-2021-3121, CVE-2020-28851, CVE-2019-11254, CVE-2020-8559, CVE-2019-11253, WS-2021-0200, CVE-2020-8565, CVE-2020-8552, CVE-2018-16886, CVE-2018-1098, CVE-2020-9283, CVE-2021-44716, CVE-2021-38561, CVE-2021-31525, CVE-2019-11254, CVE-2020-27813, CVE-2018-14042, CVE-2021-41184, CVE-2019-8331, CVE-2015-9251, CVE-2020-15113, CVE-2018-1099, CVE-2021-41183, CVE-2019-11250, CVE-2019-11253, CVE-2020-14040, CVE-2020-15106, CVE-2020-11023, CVE-2020-14040, CVE-2020-15112, CVE-2021-41182, CVE-2020-8565, CVE-2020-11022, CVE-2020-29652, CVE-2016-7103, CVE-2018-20677, Several CVE on base UBI 8 image.

Version 3.0.3

This CASE version contains IBM Netcool Operations Insight Event Integrations Operator version 3.2.3 compatible with IBM Netcool Operations Insight 1.6.3.2

1. This operator version includes a patch in the Message Bus Probe and Message Bus Gateway to resolve Log4j CVE-2021-44832 and CVE-2021-45105. The Log4j library is upgraded to 2.17.1.
    a. Message Bus probe operand image upgraded to version 18.1.3-amd64. This operand image includes Transport Module 33.4.0.
    b. Message Bus gateway operand image upgraded to version 14.2.3-amd64. This operand image includes Transport Module 33.4.0.
    c. CEM Gateway operand image upgraded to version 3.9.3-amd64. This operand image includes Jnetcool 8.4.0.

Version 3.0.2

This CASE version contains IBM Netcool Operations Insight Event Integrations Operator version 3.2.2 compatible with IBM Netcool Operations Insight 1.6.3.2

1. This operator version includes a patch in the Message Bus Probe and Message Bus Gateway to resolve Log4j CVE-2021-45046. The Log4j library is upgraded to 2.16.0.
    a. Message Bus probe operand image upgraded to version 18.1.2-amd64. This operand image includes Transport Module 33.2.0.
    b. Message Bus gateway operand image upgraded to version 14.2.2-amd64. This operand image includes Transport Module 33.2.0.

   c. CEM Gateway operand image upgraded to version 3.9.2-amd64. This operand image includes Jnetcool 8.2.0.
2. Fixes an issue with Transport Module debug logging after the upgrade to Log4j 2.15.0.

Version 3.0.1

This CASE version contains IBM Netcool Operations Insight Event Integrations Operator version 3.2.1 compatible with IBM Netcool Operations Insight 1.6.3.2

1. This operator version includes a patch in the Message Bus Probe and Message Bus Gateway to resolve Log4j CVE-2021-44228. The Log4j library is upgraded to 2.15.0.
   a. Message Bus probe operand image upgraded to version 18.1.1-amd64. This operand image includes Transport Module 33.1.0.
   b. Message Bus gateway operand image upgraded to version 14.2.1-amd64. This operand image includes Transport Module 33.1.0.
   c. CEM Gateway operand image upgraded to version 3.9.1-amd64. This operand image includes Jnetcool 8.1.0.

Version 3.0.0

This CASE version contains IBM Netcool Operations Insight Event Integrations Operator version 3.2.0 compatible with IBM Netcool Operations Insight 1.6.3.2

1. This operator version requires Openshift 4.6, 4.7 and 4.8.
2. Introduction of a new subscription channel version "v3.2" in the operator bundle. This allows users to select the operator version to install from the available subscription channels. If you have an older operator version installed, you can select the "v3.2" subscription channel to upgrade your operator to version 3.2.0.
3. The default Security Context Constraints (SCC) requirement has changed and the operator can now run with the OpenShift `restricted` SCC or define a custom SCC using `ibm-netcool-integrations-restricted-scc` as specified in the documentation. This SCC requires pods to run with an artbitrary UID. To upgrade your existing operand instances in a namespace with the `restricted` SCC, update your operands to the latest version as specified below and set the `setUIDandGID` parameter to `false`.
4. Default Kafka Probe operand version (`spec.version`) upgraded to `3.4.0`.
   a. Message Bus probe operand image upgraded to version 18.1.0. This version has several vulnerability patches.
   b. Netcool Integrations Utility image version upgraded to version 3.13.0.
5. Default Pulsar Probe operand version (`spec.version`) upgraded to `1.2.0`.
   a. Message Bus probe operand image upgraded to version 18.1.0. The default `imageTag` value is updated to `18.1.0-amd64`. This version has several vulnerability patches.
   b. Netcool Integrations Utility image version upgraded to version 3.13.0.
6. Prometheus Probe and CEM Probes operand version (`spec.version`) upgraded to `4.9.0`.
   a. Message Bus probe operand image upgraded to version 18.1.0. The default `imageTag` value is updated to `18.1.0-amd64`. This version has several vulnerability patches.
   b. Netcool Integrations Utility image version upgraded to version 3.13.0.
7. Default CEM Gateway operand version (`spec.version`) upgraded to `1.7.0`.
   a. CEM Gateway operand image upgraded to version `3.9.0-amd64`. This version has several vulnerability patches.
8. A new `probe.configPVC` parameter introduced in Kafka and Pulsar probes. This allows you pre-create a persistent volume and persistent volume claim to store your custom probe rules files and

JSON parser configuration file. Please refer to the README on how to provide custom rules files in a persistent volume and trigger the probe to reload the rules.

   a. Set the `probe.configPVC.name` parameter to the pre-created persistent volume claim name. This parameter must be set for the probe pod to mount to the PV.
   b. Set the `probe.configPVC.rulesFile` to the main rules file path in the PV. When this parameter is set, it takes precedence over `probe.rulesConfigmap`.
   c. (Optional) You can also set the `probe.configPVC.parserConfigFile` parameter to the file path of the JSON parser configuration file in the PV.

9. You can also enable common Transport Module debug logging by setting the `probe.enableTransportDebugLog` to `true` for troubleshooting purposes in Kafka, Pulsar, Prometheus and CEM probes.
10. Introduction of a new Gateway for JDBC Integration API (`JDBCGateway`) that deploys a JDBC Gateway. The gateway processes events and alerts from IBM Netcool/OMNIbus ObjectServer and forwards them to a target Relational Database Management System (RDBMS) database.
11. Fixed an issue in the `create-noi-secret.sh` script in the CASE bundle where the encrypted password file contains a blank line causing an authentication error because the secret resource created for ObjectServer secured connection contains an incorrect encrypted password string.
12. Fixed an issue where the workloads (deployment or statefulset) could not be scaled with the `kubectl scale` command or by the Horizontal Pod Autoscaler. The issue occurred because the operator reconciles the custom resource when there's an update to the workload (deployment or statefulset) resource `spec` section causing the `spec.replica` value to be reverted following the custom resource configuration.
13. Migrate operator images (ibm-netcool-integrations-operator, ibm-netcool-integrations-operator-bundle, and ibm-netcool-integrations-operator-catalog) to the IBM Container Registry (ICR) `cpopen` namespace (`icr.io/cpopen`) from Docker Hub (`docker.io/ibmcom`) for easy discoverability by customers via standard Openshift Container Platform function. These images contains Apache 2.0 license.
14. The `CEMProbe`, `PrometheusProbe` , and `CEMGateway` CustomResourceDefinition (CRD) has been updated to version `v1`. You must update the CRD in your cluster before upgrading the operator.
15. Added new Audit and Reporting mode provisioning scripts for MySQL database. These scripts are to be used to provision the tables on the database for integration with the JDBC Gateway.
16. Introduction of a new Gateway for Pulsar Integration (`PulsarGateway`) that deploys Message Bus Gateway as Pulsar producer. The gateway processes events and alerts from IBM Netcool/OMNIbus ObjectServer and forwards them to Pulsar. You can deploy a Gateway for Pulsar Integration by installing the `PulsarGateway` custom resource onto your cluster.
17. The CASE `ibmNetcoolGatewayCEMSetup` inventory has been renamed to `ibmNetcoolGatewaySetup` because the scripts provided in this inventory directory are applicable to the `CEMGateway`, `PulsarGateway` and `JDBCGateway`.
18. Upgraded PodDisruptionBudget resource API version to `v1` from `v1beta1`.
19. Added `operatorframework.io/arch.amd64: supported` and `operatorframework.io/os.linux: supported` labels into the operator ClusterServiceVersion resource. This allows OLM to identify which platform architecture and operating system is supported by the operator.
20. A `NetworkPolicy` resource is deployed with each probe or gateway instance installed to allow incoming and outgoing traffic to the pods. This can be disabled by setting the `global.enableNetworkPolicy` parameter to `false` in the custom resource.
21. The probe or gateway custom resource phase (`status.phase`) is now returned as one of the table columns when retrieving the resource using `oc get` command.
22. The CASE pre-requisite specification file now only checks if the required CLI tools are installed without restricted to a specific versions to give flexibility for users to upgrade to latest version available.

23. New configurable parameters `resources.limits.ephemeral-storage` (default is `500Mi`) and `resources.requests.ephemeral-storage` (default is `100Mi`) is introduced in all probe and gateway custom resources to configure the pod ephemeral storage request and limit.
24. New configurable parameters introduced in `PulsarProbe` API to configure Pulsar Probe:
    a. The `probe.pulsar.consumer.subscriptionType` (default is `Exclusive`) allows users to configure Pulsar Consumer subscription type parameter.
    b. The `probe.pulsar.inactivity` (default is 0) allows users to configure the inactivity period (in seconds) to initiate a shutdown when there is no incoming events received by the probe.
    c. The `probe.pulsar.maxBufferedReaderLength` (default is 16384) allows users to configure maximum length (in bytes) of the incoming message line. This can be increased to allow the probe to accept large JSON payload messages but the recommended limit should not exceed 24576 bytes (24kB).
25. This operator version resolves several CVEs detected in our operand docker image:

    CVE-2021-21409, CVE-2021-37137, CVE-2021-37136, CVE-2021-35515, CVE-2021-36090, CVE-2021-35516, CVE-2021-35517, CVE-2021-35515, CVE-2021-36090, CVE-2021-35516, CVE-2021-35517, CVE-2021-35515, CVE-2021-36090, CVE-2021-35516, CVE-2021-35517, CVE-2021-35515, CVE-2021-36090, CVE-2021-35516, CVE-2021-35517, Several CVEs from Redhat UBI 8 base image.

Version 2.1.0

1. Introduction of a new Probe for Pulsar Integration (`PulsarProbe`) that deploys the Message Bus Probe as a Pulsar consumer. The probe can connect to a Pulsar broker and subscribe JSON events from one or more Pulsar topics. You can deploy a Probe for Pulsar Integration by installing the `PulsarProbe` custom resource onto your cluster.
2. Default Kafka Probe operand version (`spec.version`) upgraded to `3.2.0`.
    a. Message Bus probe operand image upgraded to version 16.1.0. The default `imageTag` value is updated to `16.1.0-amd64`. This version has several vulnerability patches.
    b. Netcool Integrations Utility image version upgraded to version 3.7.0.
    c. Image pull policy `imagePullPolicy` default value has changed to `IfNotPresent`, previously `Always`.
3. Default Prometheus Probe and CEM Probe operand version (`spec.version`) upgraded to `4.8.0`. This operand version contains the following changes:
    a. Message Bus probe operand image upgraded to version 16.1.0. The default `imageTag` value is updated to `16.1.0-amd64`. This version has several vulnerability patches.
    b. Netcool Integrations Utility image version upgraded to version 3.7.0.
    c. Image pull policy `imagePullPolicy` default value has changed to `IfNotPresent`, previously `Always`.
4. Default CEM Gateway operand version (`spec.version`) upgraded to `1.6.0`. This operand version contains the following changes:
    a. CEM Gateway container image upgraded to version 3.3.0. The default `imageTag` value is updated to `3.2.0-amd64`. This version has several vulnerability patches.
    b. Netcool Integrations Utility image version upgraded to version 3.7.0.
    c. Image pull policy `imagePullPolicy` default value has changed to `IfNotPresent`, previously `Always`.
5. Several improvements and fixes in the Container Application Software for Enterprise (CASE):
    a. The `netcoolIntegrationsOperator` and `airgap` inventories has been merged into the `netcoolIntegrationsOperatorSetup` inventory to simplify the steps to install the operator in an disconnected environment.

b. Fixed an issue in CASE `install_catalog` launch action where the optional `--image` argument isn't captured because it overlaps with another action argument. In this launch action, the `--image` argument is replaced with `--catalogImage` allowing users to specify the catalog image repository to use in the `CatalogSource` resource. If both `--catalogImage` and `--registry` arguments are specified, then the registry specified will appended with the repository path from the `--catalogImage` argument.

c. Skopeo version 1.0.0 tool is a pre-requisite for the `mirror-images` CASE launch actions. This is to enable the CASE launch scripts to be able to copy and mirror images to the target registry.

d. Fixed an issue where some CASE launch actions pre-requisite check fails when run with Docker version 20.0. The pre-requisite check accepts Docker version 20.x.

e. The redundant createSecurityNamespacePrereqs.sh script in the `ibmNetcoolGatewayCEMSetup` inventory has been removed because the namespace Security Context Constraints check is already performed when running the `install-operator` and `install-operator-native` CASE actions.

6. The operator catalog image `ibm-netcool-integrations-operator-catalog` tag is now version specific for operator version range 1.0.0 <= v < 2.0.0, for example `1.3.0-amd64`. The `latest` image tag will be applied to the catalog image for the latest operator version version > 2.0.0.
7. Skopeo version 1.0.0 tool is a pre-requisite for the `mirror-images` CASE launch actions. This is to enable the CASE launch scripts to be able to copy and mirror images to the target registry.
8. Fixes an issue where the Promethues Probe and Kafka Probe doesn't create a Pod Disruption Budget resource when the `podisruptionbudget.enabled` parameter is set to `true`
9. Enhanced the operator to recreate the Deployment or Statefulset resource if they are deleted before the custom resource is deleted.
10. Fixed issue where catalog source isn't deleted when using CASE uninstall-catalog action command. This action now uses the namespace specified by the user.
11. Several vulnerabilities fixes included this operator and operand version:

    CVE-2016-10228, CVE-2017-14502, CVE-2019-2708, CVE-2019-3842, CVE-2019-9169, CVE-2019-13012, CVE-2019-18276, CVE-2019-25013, CVE-2020-8231, CVE-2020-8284, CVE-2020-8285, CVE-2020-8286, CVE-2020-8927, CVE-2020-9948, CVE-2020-9951, CVE-2020-9983, CVE-2020-13434, CVE-2020-13543, CVE-2020-13584, CVE-2020-13776, CVE-2020-15358, CVE-2020-24977, CVE-2020-27618, CVE-2020-28196, CVE-2020-29361, CVE-2020-29363, CVE-2021-3326, CVE-2021-3449, CVE-2021-3450, CVE-2021-3516, CVE-2021-3517, CVE-2021-3518, CVE-2021-3520, CVE-2021-3537, CVE-2021-3541, CVE-2021-20271, CVE-2021-20305, CVE-2021-27219

Version 2.0.0

1. Introduction of a new subscription channel version "v2.0" in the operator catalog. This allows users to select the operator version to install from the available subscription channels. If you have operator version 1.2.0 from the "beta" subscription channel already installed in your cluster, you can select the "v2.0" subscription channel to upgrade your operator to version 2.0. **Note** Please review the IBM Cloud Pak® for AIOps license terms before upgrading.
2. The operator controller has the capability to perform a global image upgrade to all operands. See Global Image Setting section for details.
3. Default Kafka Probe operand version (`spec.version`) upgraded to `3.1.0`. This operand version contains the following changes:

   a. This version introduces `probe.kafka.client.sasl.jaasConfigSecretName` in the `KafkaProbe` custom resource for users to configure Kafka consumer Java™ Authentication and Authorization Service (JAAS) configuration file in a Kubernetes secret resource and then mount it to the probe by setting this parameter to the secret name. When the probe pod is created, the JAAS configuration will be mounted to the probe's Kafka transport "kafka_client_jaas.conf" configuration file. This gives flexibility to users to configure their Kafka probe JAAS configuration to authenticate with their Kafka broker using authentication modules other than the plain login module. If plain login module is used, you can either set the `kafka_username` and `kafka_password` in the probe secret (`probe.secretName`) or define the full JAAS configuration file in the a secret and use the new `probe.kafka.client.sasl.jaasConfigSecretName` property. For example, to create a secret with a custom JAAS configuration, define the configuration in a file "jaas-config-file.txt" and then create a secret from it:

```
tee jaas-config-file.txt <<EOF
KafkaClient {
    org.apache.kafka.common.security.scram.ScramLoginModule required
username="<username>" password="<password>";
};
EOF

kubectl create secret generic my-jaas-config --from-file=kafka-client-jaas=jaas-config-
file.txt
```

   b. Message Bus probe operand image upgraded to version 14.0. The default `imageTag` value is updated to `14.0.0-amd64`. This version has several vulnerability patches.
   c. Netcool Integrations Utility image version upgraded to version 3.5.0.

4. Default Prometheus Probe and CEM Probe operand version (`spec.version`) upgraded to `4.7.0`. This operand version contains the following changes:
   a. Message Bus probe operand image upgraded to version 14.0. The default `imageTag` value is updated to `14.0.0-amd64`. This version has several vulnerability patches.
   b. Netcool Integrations Utility image version upgraded to version 3.5.0.
5. Default CEM Gateway operand version (`spec.version`) upgraded to `1.5.0`. This operand version contains the following changes:
   a. CEM Gateway container image upgraded to version 3.1.0. The default `imageTag` value is updated to `3.1.0-amd64`. This version has several vulnerability patches.
   b. Netcool Integrations Utility image version upgraded to version 3.5.0.
6. This operator version resolves several CVEs detected in our operand docker image:

   CVE-2020-25649, CVE-2020-24659, CVE-2020-1971, CVE-2019-17571, CVE-2017-9735, CVE-2017-7656, CVE-2017-7657, CVE-2017-7658
7. The version parameter (`spec.version`) of each custom resource is a mandatory parameter and must be set in the custom resource YAML. You can use the default operand version provided in the custom resource YAML or set this parameter to the desired operand version.
8. Removed redundant `prometheusProbe.enabled` in PrometheusProbe custom resource as this parameter must be set to `true` in the custom resource.
9. Removed redundant `cemProbe.enabled` in CEMProbe custom resource as this parameter must be set to `true` in the custom resource.
10. Added `olm.relatedImage` annotations in operator deployment template to store the current operand image repository and tag.
11. Operator adds an additional policy rule into the generated `Role` resource to allow the ServiceAccount to use the WAIOps restricted Security Context Constraint (`aiops-restricted`) installed by AIOps Orchestrator operator.
12. Fixes an issue where the generated service account is not correctly set in the pod specification when `global.serviceAccountName` parameter is unset.

Global settings

The operator is now configured to perform an upgrade to all operand images by default. This is controlled by the `UPDATE_IMAGES_FROM_ENV` environment variable in the controller deployment resource and the default value is `true`. Several new annotations (`olm.relatedImage.*`) and environment variables (`IMAGE_NOIINT_*`) are introduced to provide the default images used by the controller. The controller will update all operand container resources with image value set in the `olm.relatedImage.*` annotations.

If you would like to change the global image version in all installed operands, you can opt update the respective `olm.relatedImage.*` annotation in the operator `ClusterServiceVersion` resource with the new image repository value and wait for the operator controller pod to restart for the new settings to take affect.

To disable the global image upgrade functionality, change the `UPDATE_IMAGES_FROM_ENV` environment variable to `false` in the operator `ClusterServiceVersion` resource and wait for the operator pod to restart. When disabled, the controller will use the image settings from the operand custom resources YAML and update all operand containers accordingly.

Note: It is not recommended to modify the operator `ClusterServiceVersion` resource unless necessary such as to apply a container image patch.

Version 1.2.0

Release date: 11 December 2020 Part number: CC8YGML

- Red Hat Openshift Container Platform (OCP) 4.6 support. This operator version is verified running on OCP 4.5 and 4.6
- New `KafkaProbe` kind which deploys the Message Bus probe as a Kafka consumer. You can configure this probe to integrate with Kafka systems such as IBM Event Streams on IBM Cloud.
- Several changes in the Operator cluster service version file to add links to specific documentation sections in the description for easier navigation and updated provided API name and description include a license file link in each CRD tile description.
- Add `app.kubernetes.io/part-of:` label into operator deployment resource.
- License file updated with NOI 1.6.3 license (L/N: L-TKAI-BTYDF9).

IBM Netcool Operations Insight v1.6.3 release notice: https://www.ibm.com/support/pages/node/6377824

CVE patches

This version resolves the following CVEs detected in our operator and operand docker image:

`CVE-2020-26160`, `CVE-2020-13956`, `CVE-2012-5783`, `CVE-2015-5262`

Version 1.1.0

Release date: 30 October 2020 Part number: CC8A7ML

- Supported on Red Hat Openshift Container Platform (OCP) 4.4 and 4.5.
- New CASE launcher script to assist mirroring image in an internal registry for offline (air-gap) installation or in a restricted network.
- Operator version 1.1.0 includes the following changes:
  1. [FIX] Missing parameter when using OLM UI to create operand instances.
  2. Prometheus Probe and CEM Probe version 4.5.0 support. This version includes an updated probe and utility images which resolves CVEs listed below.
  3. CEM Gateway version 1.3.0 support. This version includes an updated probe and utility images which resolves CVEs listed below.

IBM Netcool Operations Insight v1.6.2 release notice: https://www.ibm.com/support/pages/node/6243794

CVE patches

This version resolves the following CVEs detected in our operator and operand docker image:

`CVE-2020-13777`, `CVE-2020-12049`, `CVE-2020-11080`

Version 1.0.1

Release date: 07 August 2020 Part number: CC7LBML

- Supported on Red Hat Openshift Container Platform (OCP) 4.3 and 4.4.
- [FIX] OLM unable to retrieve bundle information from operator bundle image.

IBM Netcool Operations Insight v1.6.1 release notice: https://www.ibm.com/support/pages/node/6221238

Version 1.0.0

Release date: 30 June 2020 Part number: CC714ML

Initial release.

- Supported on Red Hat Openshift Container Platform (OCP) 4.3 and 4.4.
- You can deploy Prometheus Alert Manager on OCP by creating a `prometheusprobe` or custom resource.

IBM Netcool Operations Insight v1.6.1 release notice: https://www.ibm.com/support/pages/node/6221238

Prerequisites

This section describes the prerequisites to installing and deploying the IBM Netcool Operations Insight Event Integrations Operator.

The IBM Netcool Operations Insight Event Integrations Operator requires IBM Tivoli Netcool/OMNIbus ObjectServer to be created and running on Red Hat OpenShift Container Platform (OCP) 4.8, 4.9 or 4.10. IBM Netcool Operations Insight 1.6.7 is required to create IBM Tivoli Netcool/OMNIbus ObjectServer on OCP 4.8, 4.9 or 4.10. Refer to the IBM Netcool Operations Insight documentation: https://www.ibm.com/docs/en/noi

Prerequisites for Probe for Prometheus Integration

- Scope-based Event Grouping automation installed, see the following topic in IBM Documentation: About scope-based event grouping. The probe requires several table fields to be installed in the ObjectServer. For on-premise installation, see Installing scope-based event grouping. The events will be grouped by a preset ScopeId in the probe rules file if the event grouping automation triggers are enabled.
- Prometheus 2.3.1 and Prometheus Alert Manager 0.15.0 or higher

Resources required

This section describes the resources that each probe or gateway require per pod.

The operator controller pod resource requirements is as below:

- CPU:
  - Requested 100m (100 millicpu)
  - Limit 200m (200 millicpu)
- Memory
  - Requested 64Mi (~ 67 MB)
  - Limit 256Mi (~ 267 MB)

  Typically each probe or gateway pod requires at least the following level of resources per pod:
- CPU Requested : 100m (100 millicpu)

- Memory Requested : 128Mi (~ 134 MB)

You can increase or decrease the resource request and limit by modifying the resources settings (`resources.limits` and `resources.requests`) in the customer resource YAML.

Cluster configuration

The minimum cluster configuration required is 1 master node and 3 worker nodes. The recommended cluster configuration is 3 master nodes and 5 worker nodes. This operator requires worker nodes that support amd64 architecture.

Ports information and firewall requirements

This section describes which ports need to be exposed to allow clients to connect to them.

Server type integrations such as the Probe for Prometheus (`PrometheusProbe`) can be configured to expose the service which may require your firewall administrator to open certain ports for clients to connect to them.

Client type integrations such as the Probe for Kafka Integration (`KafkaProbe`), Probe for Pulsar Integration (`PulsarProbe`), and Gateway for JDBC Integration (`JDBCGateway`) and Gateway for Pulsar Integration (`PulsarGateway`) does not require ports to be exposed.

Table 2. Ports and protocols required to allow ingress communication

From	To	Ports	Function	Integration Kind
External HTTP event sources	Probe webhook	443 / 80	Send HTTP(s) POST notifications to the probe.	`WebhookProbe` and `PrometheusProbe`

Each probe and gateway instances creates a `NetworkPolicy` resource to allow ingress and egress traffic to the pods by default. You can list the `NetworkPolicy` resources using the `oc get networkpolicy` command.

Security context constraints

On Red Hat OpenShift Container Platform (OCP) this operator requires a SecurityContextConstraints resource to be bound to the target namespace prior to installation.

The predefined SecurityContextConstraints name: `ibm-restricted-scc` has been verified for this operator, if your target namespace is bound to this SecurityContextConstraints resource you can proceed to install the operator.

Alternatively, for this operator you can define a custom SecurityContextConstraints resource which can be used to finely control the permissions and/or capabilities needed to deploy the operator. You can enable this custom SecurityContextConstraints resource using the command line tool.

- Custom `SecurityContextConstraints` definition:

```
apiVersion: security.openshift.io/v1
kind: SecurityContextConstraints
metadata:
  annotations:
    kubernetes.io/description: restricted denies access to all host features and requires
      pods to be run with a UID, and SELinux context that are allocated to the namespace.
This
      is the most restrictive SCC and it is used by default for authenticated users.
  name: ibm-netcool-integrations-restricted-scc
allowHostDirVolumePlugin: false
allowHostIPC: false
allowHostNetwork: false
allowHostPID: false
allowHostPorts: false
allowPrivilegeEscalation: true
```

```
allowPrivilegedContainer: false
allowedCapabilities: null
defaultAddCapabilities: null
fsGroup:
  type: MustRunAs
priority: null
readOnlyRootFilesystem: false
requiredDropCapabilities:
- KILL
- MKNOD
- SETUID
- SETGID
runAsUser:
  type: MustRunAsRange
seLinuxContext:
  type: MustRunAs
supplementalGroups:
  type: RunAsAny
users: []
volumes:
- configMap
- downwardAPI
- emptyDir
- persistentVolumeClaim
- projected
- secret
```

Gathering information for IBM Support

This topic describes how to gather information for IBM Support.

To help IBM Support to troubleshoot any issues observed in your IBM Netcool Operations Insight Event Integration instance, use the following procedure to collect logs and information from your cluster:

1. Log in to your Red Hat OpenShift Container Platform by using the oc CLI. You must login as a namespace administrator or a cluster administrator who has permissions to read resources in the target namespace.
2. Set the `namespace` variable to the namespace where the instance is deployed:

   ```
   export namespace=my-namespace
   ```

3. Get the instance description in your namespace:

   a. To get a specific instance resource description, run the following command:

      ```
      kubectl get
      prometheusprobe,kafkaprobe,webhookprobe,pulsargateway,jdbcgateway,kafkagateway --
      namespace $namespace
       kubectl describe <instance kind>/<instance name> --namespace $namespace
      ```

      Replace `<instance kind>` and `<instance name>` in the second command with the resource name from the output from the first command.

   b. Or run the following command to get all instance descriptions in the namespace and capture the command output:

      ```
      for i in $(kubectl get
      prometheusprobe,kafkaprobe,webhookprobe,pulsargateway,jdbcgateway,kafkagateway --no-
      headers --namespace
      $namespace | awk '{print $1}'); do kubectl describe $i ; done
      ```

4. Get the pod names of the instance:

   ```
   kubectl get pods -l app.kubernetes.io/instance=example-webhookprobe --namespace $namespace
   ```

5. Using the pod names from the previous command, run the following command to get the instance pod logs and capture the command output:

   ```
   kubectl logs <pod name>
   ```

If there is more than one pod, run the same command for each pod name.

6. Capture the operator `deployment` resource description:

```
kubectl describe deployment netcool-integrations-operator --namespace $namespace
```

7. Capture the operator pod logs:

```
kubectl get pods -l app.kubernetes.io/instance=netcool-integrations-operator  --namespace
$namespace
kubectl logs <operator-pod> --namespace $namespace
```

Replace `<operator-pod>` in the command with the name of the operator pod from the `kubectl get pods` output.

8. Get the list of `ConfigMap` used by your instance and its data:

```
kubectl get configmap --namespace $namespace
kubectl kubectl describe <configmap> --namespace $namespace
```

Replace `<configmap>` with the `ConfigMap` name from the `kubectl get configmap` output. You should provide each `ConfigMap` data for troubleshooting.

To help IBM Support to troubleshoot any issues observed in your IBM Netcool Operations Insight Event Integration instance, follow the procedure below to collect logs and information from your cluster.

a. Log in to your Red Hat OpenShift Container Platform by using the `oc` CLI. You must login as a namespace administrator or cluster administrator who has permissions to read resources in the target namespace.
b. Set the `namespace` variable with the namespace where the instance is deployed.

```
export namespace=my-namespace
```

Role-based access control

Role-Based Access Control (RBAC) is applied to the operator by using a custom service account having a specific role binding. RBAC provides greater security by ensuring that the it operates within the specified scope.

The following are the options to apply RBAC to the operator:

- "Deploying the operator with pre-created role and role binding resources" on page 18: The Cluster Administrator per-creates the role and role binding resources prior to deploying the operator. With the role binding resources in place, the operator may be assigned to another user such as an Administrator to perform the deployment. This option may require additional setup prior to setting up the operator, but it enables the RBAC resources to be easily applied to multiple operator deployments in the namespace.
- "Deploying the operator with the role and role binding resources included with the operator" on page 20: The Cluster Administrator deploys the operator with the role and role binding resources being included with the operator. This option automates the creation of the RBAC resources for the operator deployment, but the operator deployment must be performed by the Cluster Administrator. The RBAC resources is managed by the operator and will be removed when the release is deleted.

Deploying the operator with pre-created role and role binding resources

The Cluster Administrator will apply these steps to pre-create the required RBAC resources, namely: service account, role and role bindings, before the operator is deployed. Once the RBAC resources are created, the operator deployment may be done by another user for example: `Administrator`.

1. As the Cluster Administrator, create a YAML file called `serviceAccount.yaml` with the contents below, where `Release_Name` is the release name of the operator, `Namespace` is the target namespace, `Probe_Name` is the probe name and `Secret_Name` is the image pull Secret.

   ```
   apiVersion: v1
   kind: ServiceAccount
   metadata:
       name: <Release_Name>-ibm-netcool-probe-sa
       namespace: <Namespace>
       labels:
           app.kubernetes.io/name: "<Probe_Name>"
           helm.sh/chart: "<Probe_Name>"
           app.kubernetes.io/instance: "<Release_Name>"
           release: "<Release_Name>"
           app.kubernetes.io/component: "common"
   imagePullSecrets:
   - name: <Secret_Name>
   ```

   Create the Service Account using the following command:

   ```
   kubectl apply -f serviceAccount.yaml
   ```

2. As the Cluster Administrator, create a YAML file called `role.yaml` with the contents below, where `Release_Name` is the release name of the operator and `Namespace` is the target namespace.

   ```
   kind: Role
   apiVersion: rbac.authorization.k8s.io/v1
   metadata:
       name: <Release_Name>-ibm-netcool-probe-role
       namespace: <Namespace>
       labels:
           app.kubernetes.io/name: "<Probe_Name>"
           helm.sh/chart: "<Probe_Name>"
           app.kubernetes.io/instance: "<Release_Name>"
           release: "<Release_Name>"
           app.kubernetes.io/component: "common"
   rules:
   - apiGroups: [""] # "" indicates the core API group
       resources: ["secrets"]
       verbs: ["get", "list"]
   ```

   Create the Role using the following command:

   ```
   kubectl apply -f role.yaml
   ```

3. As the Cluster Administrator, create a YAML file called `roleBinding.yaml` with the contents below, where `Release_Name` is the release name of the operator and `Namespace` is the target namespace.

   ```
   kind: RoleBinding
   apiVersion: rbac.authorization.k8s.io/v1
   metadata:
       name: <Release_Name>-ibm-netcool-probe-rolebinding
       namespace: <Namespace>
       labels:
           app.kubernetes.io/name: "<Probe_Name>"
           helm.sh/chart: "<Probe_Name>"
           app.kubernetes.io/instance: "<Release_Name>"
           release: "<Release_Name>"
           app.kubernetes.io/component: "common"
   subjects:
   - kind: ServiceAccount
       name: <Release_Name>-ibm-netcool-probe-sa
   namespace: <Namespace>
   roleRef:
       kind: Role
       name: <Release_Name>-ibm-netcool-probe-role
       apiGroup: rbac.authorization.k8s.io
   ```

   Create the Role Binding using the following command:

   ```
   kubectl apply -f roleBinding.yaml
   ```

4. Set the service account name created earlier in the operator's **`global.serviceAccountName`** parameter. As an Administrator, edit `values.yaml` and update the `serviceAccountName` with the name of the Service Account created earlier.

   ```
   global:
       image:
           secretName: "<Secret_Name>"
       serviceAccountName: "<Release_Name>-ibm-netcool-probe-sa"
   ```

5. Perform all required edits to `values.yaml` and deploy the operator.

Deploying the operator with the role and role binding resources included with the operator

When the Cluster Administrator applies these steps, the required RBAC resources ie. service account, role and role bindings will be created when the operator is deployed. It should be noted that the same RBAC resources will be removed automatically when the Helm release is deleted.

1. As the Cluster Administrator, edit `values.yaml` and specify the image pull Secret for the `secretName` parameter. Update the `serviceAccountName` parameter to be blank. These settings enable the operator to automatically create the require service account, role and role bindings using the specified Secret to pull the image.

   ```
   global:
       image:
           secretName: "<Secret_Name>"
       serviceAccountName: ""
   ```

2. Perform all required edits to `values.yaml` and deploy the operator.

Operator scope

This IBM Netcool Operations Insight Event Integrations Operator is a namespaced-scoped operator that watches a single namespace where the operator is deployed. To deploy probes or gateways on more than one namespace, it is recommended to deploy a separate operator to watch the namespace.

The operator supports the following installation modes when installing using **Operator Lifecycle Manager** from the **Web Console**:

- **All namespaces mode**

  The operator controller will be installed in the openshift-operators namespace and watches all namespaces on the cluster for custom resources. This means that the operator can be installed only once in this mode. There can only exist a single version of the operator controller on the cluster when installed in this mode. To run in this mode, the operator role based access control (RBAC) rules will be elevated from `Role` to `ClusterRole` so that the Operator controller can manage operands running on all namespaces.

- **Single namespace mode (Own namespace)**

  The operator controller and the operands it watches are isolated to a single namespace of your choice. This means that a product operator can be installed multiple times, across multiple namespaces, in the same cluster at different versions if it is completely isolated in a single namespace. Although it is possible for multiple instances of an operator to be installed on a cluster, it is not recommended.

Storage

Gateways uses persistent volume to store persistent data such as the cache file for Store-and-Forward functionality that allows the gateway process to resume data processing from the last data fetched and forward it to the Object Server.

Probes that support persistent custom rules files and JSON parser configuration use a persistent volume for users to provide a custom rules file with a JSON parser configuration file that is mounted to the probe pods for event processing. The persistent volume is not required by default and only required if users enable the persistent rules files when configuring the probe.

Table 3. Integration types and access modes required

Integration type	Access modes required
Gateway for JDBC Integration	ReadWriteMany (RWX) or ReadOnlyMany (ROX)
Gateway for Kafka Integration	ReadWriteOnce (RWO)
Gateway for Pulsar Integration	Not applicable. Store-and-Forward is not supported.
Probe for Pulsar Integration	ReadWriteMany (RWX)
Probe for Kafka Integration	ReadWriteMany (RWX)
Probe for Webhook Integration	ReadWriteMany (RWX)
Probe for Prometheus Integration	Not applicable.

Each integration supports the following provisioning modes:

- **Dynamic provisioning**

  For storage providers that support dynamic provisioning, you can specify the storage class name in the probe or gateway custom resource to request persistent volume from the storage class specified or let the default storage class to provision a volume for your pods.
- **Static provisioning**

  For static provisioning, you can also specify labels to bind to specific storage volume.

The probes and gateways has been verified with Rook Ceph® storage provider.

Installing the IBM Netcool Operations Insight Event Integrations Operator

The following tasks represent the typical task flow for performing your IBM Netcool Operations Insight Event Integrations Operator installation:

“1. Install Netcool Operations Insight” on page 21

“2. Configure Storage” on page 22

“3. Add the operator catalog and prepare your cluster” on page 22

“4. Install IBM Netcool Operations Insight Event Integrations Operator” on page 24

“5. Attach entitlement key” on page 25

“6. Create your integration instance” on page 25

1. Install Netcool Operations Insight

The IBM Netcool Operations Insight Event Integrations Operator requires IBM Tivoli Netcool/OMNIbus ObjectServer to be created and running on Red Hat OpenShift Container Platform (OCP). A current version of IBM Netcool Operations Insight is required to create IBM Tivoli Netcool/OMNIbus ObjectServer on OCP. Refer to the IBM Netcool Operations Insight documentation: https://www.ibm.com/docs/en/noi

2. Configure Storage

You must define storage classes in Red Hat OpenShift Container Platform and set your storage configuration to satisfy your sizing requirements.

Gateways uses persistent volume to store persistent data such as the cache file for Store-and-Forward functionality that allows the gateway process to resume data processing from the last data fetched and forward it to the Object Server.

Probes that support persistent custom rules files and JSON parser configuration use a persistent volume for users to provide a custom rules file with a JSON parser configuration file that is mounted to the probe pods for event processing. The persistent volume is not required by default and only required if users enable the persistent rules files when configuring the probe.

Table 4. Integration types and access modes required

Integration type	**Access modes required**
Gateway for JDBC Integration	ReadWriteMany (RWX) or ReadOnlyMany (ROX)
Gateway for Kafka Integration	ReadWriteOnce (RWO)
Gateway for Pulsar Integration	Not applicable. Store-and-Forward is not supported.
Probe for Pulsar Integration	ReadWriteMany (RWX)
Probe for Kafka Integration	ReadWriteMany (RWX)
Probe for Webhook Integration	ReadWriteMany (RWX)
Probe for Prometheus Integration	Not applicable.

Each integration supports the following provisioning modes:

- **Dynamic provisioning**

  For storage providers that support dynamic provisioning, you can specify the storage class name in the probe or gateway custom resource to request persistent volume from the storage class specified or let the default storage class to provision a volume for your pods.
- **Static provisioning**

  For static provisioning, you can also specify labels to bind to specific storage volume.

The probes and gateways has been verified with Rook Ceph storage provider.

3. Add the operator catalog and prepare your cluster

To install an operator, you must first install the catalog source into OCP and then install the operator.

You can get the IBM Netcool Operations Insight Event Integrations Operator from one of the following catalogs:

- IBM Operator catalog
- IBM Netcool Operations Insight Event Integrations Operator catalog

Installing the IBM Operator catalog into OCP

1. Log in to your Red Hat OpenShift Container Platform cluster.
2. Click **Home** > **API Explorer**, then search for **CatalogSource**.
3. Click **Create CatalogSource**.

4. In the **CatalogSource name** field, enter `ibm-operator-catalog`.
5. In the **Publisher name** field, enter `IBM`.
6. In the **Image (URL of container image)** field, enter `icr.io/cpopen/ibm-operator-catalog:latest`.
7. Under **Availability**, select the **Cluster-wide CatalogSource** option.

Installing the IBM Netcool Operations Insight Event Integrations Operator catalog into OCP

1. Log in to your Red Hat OpenShift Container Platform cluster.
2. Click **Home** > **API Explorer**, then search for **CatalogSource**.
3. Click **Create CatalogSource**.
4. In the **CatalogSource name** field, enter `ibm-netcool-integrations-operator-catalog`.
5. Provide a catalog source name and the image URL:

   ```
   icr.io/cpopen/ibm-operator-catalog:latest
   ```

   If a specific version is required, change the image tag with the specific version or use the image digest.

   Attention: The dot character (`.`) is not allowed in the catalog source name.

   Select the **Create** button.
6. The catalog source appears, after a few minutes refresh the screen to see the number of operators count become 1.

CLI enablement

The catalog can be added by applying the following YAML file to the Red Hat OpenShift Container Platform cluster. Create this file and name it `catalog_source.yaml`.

```
apiVersion: operators.coreos.com/v1alpha1
kind: CatalogSource
metadata:
  name: ibm-netcool-integrations-operator-catalog
  namespace: openshift-marketplace
spec:
 displayName: IBM Netcool Operations Insight Event Integrations Catalog
 publisher: IBM
 sourceType: grpc
 image: icr.io/cpopen/ibm-operator-catalog:latest
 updateStrategy:
    registryPoll:
       interval: 45m
```

Apply the YAML file by using the command:

```
oc apply -f catalog_source.yaml -n openshift-marketplace
```

Verify the CLI installation

```
oc get CatalogSources ibm-netcool-integrations-operator-catalog -n openshift-marketplace
```

You receive this output on success:

```
NAME                                         DISPLAY                                     TYPE
PUBLISHER    AGE
ibm-netcool-integrations-operator-catalog    IBM Netcool Integrations Operator Catalog   grpc
IBM          38s
```

You receive this output on error:

```
Error from server (NotFound): catalogsources.operators.coreos.com "ibm-netcool-integrations-
operator-catalog" not found
```

4. Install IBM Netcool Operations Insight Event Integrations Operator

You can install your operator by using the OpenShift console, the OpenShift CLI, or by using the OpenShift CLI natively.

Option 1: Install the operator using the OpenShift console

To install the IBM Netcool Operations Insight Event Integrations Operator using the OpenShift console, use the following steps:

1. Log in to your OpenShift cluster's console.
2. Click **Operators > OperatorHub**. The OperatorHub page is displayed.
3. In the All Items field, enter the IBM Netcool Operations Insight Event Integrations Operator. The operator is displayed.
4. Click the IBM Netcool Operations Insight Event Integrations Operator tile.
5. Click Install. The Install Operator page is displayed.
6. Enter the following values:
   - Set **Update Channel** to the latest version available
   - Set the **Installation mode** to `A specific namespace` on the cluster.
   - Set the **Namespace** to the project (namespace) value that you want to install the operator into.

     Note: When using the console to install the operator, you can either use an existing namespace, the default namespace that is provided by the operator, or create a new namespace. If you want to create a new namespace, you can create it from this form. If you want to create a namespace before installing, from the Project list, select Create Project, specify the Name of the project that you want to create, and click Create.
   - Set **Approval** Strategy to `Automatic`.
7. Click **Install** and wait for the your operator to install.

OpenShift informs you that the installation is complete. You can verify that installation by navigating to Operators > Installed Operators, and selecting your project from the Projects dropdown. The operator that you have just installed and all of its dependent operators in the project are listed with a status of Succeeded.

Option 2: Install the operator using the OpenShift CLI

1. Determine whether you want to create a new, or reuse an existing namespace to install your operator into. To create a new namespace in the CLI, run the following command:

   ```
   oc create namespace <namespace>
   ```

   Where `<namespace>` is the name of the namespace that you want to create.
2. Create Operator Group.

   You must create an operator group in your custom project (namespace), or your operator will not install. There might be an operator group for managing a namespace for given APIs. If there is an Operator group for the namespace, do not create a second one.

   Create the Operator group by running the following command:

   ```
   cat << EOF | oc apply -f -
   apiVersion: operators.coreos.com/v1
   kind: OperatorGroup
   ```

```
metadata:
  name: ibm-netcool-integrations-operator-catalog-group
  namespace: <namespace>
spec:
  targetNamespaces:
    - <namespace>
EOF
```

3. Install your operator with the following command:

   ```
   cat << EOF | oc apply -f -
   apiVersion: operators.coreos.com/v1alpha1
   kind: Subscription
   metadata:
     name: ibm-netcool-integrations-operator-group-subscription
     namespace: <namespace>
   spec:
     channel: <latest-version-of-the-operator>
     installPlanApproval: Automatic
     name: ibm-netcool-integrations-operator-catalog-group
     source: ibm-netcool-integrations-operator-catalog
     sourceNamespace: openshift-marketplace
   EOF
   ```

   Note: Change the `source` attribute to follow the name of the catalog source installed in previous step. For example: `ibm-operator-catalog` or `ibm-netcool-integrations-operator-catalog`.

4. After a few minutes, the operator is installed. Run the following command to verify that all of the components are in the `Succeeded` state:

   ```
   oc get csv -n <namespace> | grep netcool-integrations-operator
   ```

   Where `<namespace>` is `openshift-operators` if you are using the `AllNamespaces` installation mode, or the project (namespace) if you are using the `OwnNamespace` installation mode.

5. Attach entitlement key

Check your entitlement and get an entitlement key from the My IBM Container Library website: https://myibm.ibm.com/products-services/containerlibrary. You need to store the entitlement key on your cluster by creating an Image Pull Secret. Using a Kubernetes secret allows you to securely store the entitlement key on your cluster and assign it to the operator and the operand deployments to pull images from entitled registry.

Run the following command to create the secret and add it to your OpenShift namespace:

```
kubectl create secret docker-registry ibm-entitlement-key --docker-server=cp.icr.io --docker-
username=cp --docker-password=<KEY> --docker-email=<EMAIL> --namespace <NAMESPACE>
```

Where:

`<KEY>` is your entitlement key.

`<EMAIL>` is your email address.

`<NAMESPACE>` is the name of your target OpenShift Container Platform namespace where the operator will be deployed.

The name of the secret that you are creating is `ibm-entitlement-key` with `cp` as the username and `cp.icr.io` as the registry server. The secret name is used by the operator to deploy the offering in later steps. If you change the name of any of secrets that you create, you need to change the corresponding name in later steps.

6. Create your integration instance

This section describes how to configure a probe or gateway instance either using the Openshift Console:

1. Log in to the Red Hat OpenShift Container Platform web console with your Red Hat OpenShift Container Platform cluster admin credentials.
2. Change **Project** to the namespace where you installed the operator in. Select the namespace from the **Project** drop-down list.
3. In the navigation panel, click Operators > Installed Operators.
4. In the list on the Installed Operators panel, find and click IBM Netcool Operations Insight Event Integrations Operator.
5. Click the tile or tab corresponding to the probe or gateway that you want to create and click **Create instance**.
6. The instance creation panel opens, and offers two methods for configuring the resource; the Form view and the YAML view. Select the YAML view.
7. Change the `spec.license.accept` to `true` if you accept the license agreement.
8. Configure the probe or gateway instance parameters. See "Connecting with IBM Cloud Pak for AIOps" on page 61.
9. Click **Create**. The creation process can take a longer amount of time depending on your environment.

Your instance of Cloud Pak is added to the list of instances in the current project (namespace).

Installing the IBM Netcool Operations Insight Event Integrations Operator to a private container registry

If your cluster is not connected to the internet, you can install the IBM Netcool Operations Insight Event Integrations Operator in your cluster by using connected or disconnected mirroring.

If you have a host that can access both the internet and your mirror registry, but not your cluster nodes, you can directly mirror the content from that machine. This process is referred to as **connected mirroring**. If you have no such host, you must mirror the images to a file system and then bring that host or removable media into your restricted environment. This process is referred to as **disconnected mirroring**.

Before you begin

You must complete the steps in the following sections before you begin generating mirror manifests:

- Prerequisites
- Prepare a host
- Set environment variables and download CASE files

Important: If you intend on installing using a private container registry, your cluster must support ImageContentSourcePolicy (ICSP).

Prerequisites

Regardless of whether you plan to mirror the images with a bastion host or to the file system, you must satisfy the following prerequisites:

- Red Hat® OpenShift® Container Platform requires you to have cluster admin access to run the `install-operator` command.
- An Red Hat® OpenShift® Container Platform cluster must be installed.

Note: IBM images located on Quay and Docker are being migrated to anonymous locations on icr.io. The following sites and ports might not be required depending on the version of software you are installing.

- Access to the following sites and ports:
  - `*.docker.io` and `*.docker.com`: For more information about specific sites to allow access to, see Docker Hub Hosts for Firewalls and HTTP Proxy Servers)

- `icr.io:443` for IBM Cloud Container Registry, CASE OCI artifact, and IBM Cloud Pak foundational services catalog source
- `github.com` for CASEs and tools

Tip: With `ibm-pak` plug-in version 1.2.0, you can eliminate the port for `github.com` to retrieve CASES and tooling by configuring the plug-in to download CASEs as OCI artifacts from IBM Cloud Container Registry (ICCR):

```
oc ibm-pak config repo 'IBM Cloud-Pak OCI registry' -r oci:cp.icr.io/cpopen --enable
```

Prepare a host

If you are in an air-gapped environment, you must be able to connect a host to the internet and mirror registry for connected mirroring or mirror images to file system which can be brought to a restricted environment for disconnected mirroring.

The following table explains the software requirements for mirroring the IBM Netcool Operations Insight Event Integrations Operator images:

Table 5. Software requirements and purpose

Software	Purpose
Docker	Container management
Podman	Container management
Red Hat OpenShift CLI (oc)	Red Hat OpenShift Container Platform administration

Complete the following steps on your host:

1. Install Docker or Podman.

   To install Docker (for example, on Red Hat® Enterprise Linux®®), run the following commands:

   Note: If you are installing as a non-root user you must use sudo. For more information, refer to the Podman or Docker documentation for installing as a non-root user.

   ```
   yum check-update
   yum install docker
   ```

   To install Podman, see Podman Installation Instructions.
2. Install the oc Red Hat® OpenShift® Container Platform CLI tool.
3. Download and install the most recent version of `IBM Catalog Management Plug-in for IBM Netcool Event Integrations Operator` from the IBM/ibm-pak. Extract the binary file by entering the following command:

   ```
   tar -xf oc-ibm_pak-linux-amd64.tar.gz
   ```

   Run the following command to move the file to the `/usr/local/bin` directory:

   Note: If you are installing as a non-root user you must use sudo. For more information, refer to the Podman or Docker documentation for installing as a non-root user.

   ```
   mv oc-ibm_pak-linux-amd64 /usr/local/bin/oc-ibm_pak
   ```

   Note: Download the plug-in based on the host operating system. You can confirm that `oc ibm-pak -h` is installed by running the following command:

   ```
   oc ibm-pak --help
   ```

   The plug-in usage is displayed.

For more information on plug-in commands, see command-help.

The plug-in is also provided in a container image `cp.icr.io/cpopen/cpfs/ibm-pak:TAG` where TAG should be replaced with the corresponding plugin version, for example `cp.icr.io/cpopen/cpfs/ibm-pak:v1.2.0` will have v1.2.0 of the plugin.

The following command will create a container and copy the plug-ins for all the supported platforms in a directory, plugin-dir. You can specify any directory name and it will be created while copying. After copying, it will delete the temporary container. The plugin-dir will have all the binaries and other artifacts you find in a Github release and repo at IBM/ibm-pak.

```
id=$(docker create cp.icr.io/cpopen/cpfs/ibm-pak:TAG - )
docker cp $id:/ibm-pak-plugin plugin-dir
docker rm -v $id
cd plugin-dir
```

Create registry namespaces

Top-level namespaces are the namespaces which appear at the root path of your private registry. For example, if your registry is hosted at `myregistry.com:5000`, then `mynamespace` in `myregistry.com:5000/mynamespace` is defined as a top-level namespace. There can be many top-level namespaces.

When the images are mirrored to your private registry, it is required that the top-level namespace where images are getting mirrored already exists or can be automatically created during the image push. If your registry does not allow automatic creation of top-level namespaces, you must create them manually.

When you generate mirror manifests, you can specify the top-level namespace where you want to mirror the images by setting TARGET_REGISTRY to `myregistry.com:5000/mynamespace` which has the benefit of needing to create only one namespace `mynamespace` in your registry if it does not allow automatic creation of namespaces.

If you do not specify your own top-level namespace, the mirroring process will use the ones which are specified by the CASEs. For example, it will try to mirror the images at `myregistry.com:5000/cp`, `myregistry.com:5000/cpopen etc`.

So if your registry does not allow automatic creation of top-level namespaces and you are not going to use your own during generation of mirror manifests, then you must create the following namespaces at the root of your registry.

- cp
- cpopen

There can be more top-level namespaces that you might need to create. See section on Generate mirror manifests for information on how to use the `oc ibm-pak describe` command to list all the top-level namespaces.

Set environment variables and download CASE files

If your host must connect to the internet via a proxy, you must set environment variables on the machine that accesses the internet via the proxy server.

If you are mirroring via connected mirroring, set the following environment variables on the machine that accesses the internet via the proxy server:

```
export https_proxy=http://proxy-server-hostname:port
export http_proxy=http://proxy-server-hostname:port

# Example:
export https_proxy=http://server.proxy.xyz.com:5018
export http_proxy=http://server.proxy.xyz.com:5018
```

Before mirroring your images, you can set the environment variables on your mirroring device, and connect to the internet so that you can download the corresponding CASE files. To finish preparing your host, complete the following steps:

Note: Save a copy of your environment variable values to a text editor. You can use that file as a reference to cut and paste from when you finish mirroring images to your registry.

1. Create the following environment variables with the installer image name and the version.

   ```
   export CASE_NAME=<YOUR_CASE_NAME>
   export CASE_VERSION=<YOUR_CASE_VERSION>
   ```

   To find the CASE name and version, see IBM: Product CASE to Application Version.
2. Connect your host to the intranet.
3. The plug-in can detect the locale of your environment and provide textual helps and messages accordingly. You can optionally set the locale by running the following command:

   ```
   oc ibm-pak config locale -l LOCALE
   ```

   where LOCALE can be one of de_DE, en_US, es_ES, fr_FR, it_IT, ja_JP, ko_KR, pt_BR, zh_Hans, zh_Hant.
4. Configure the plug-in to download CASEs as OCI artifacts from IBM Cloud Container Registry (ICCR).

   ```
   oc ibm-pak config repo 'IBM Cloud-Pak OCI registry' -r oci:cp.icr.io/cpopen --enable
   ```

5. Enable color output (optional with v1.4.0 and later)

   ```
   oc ibm-pak config color --enable true
   ```

6. Download the image inventory for your IBM Netcool Operations Insight Event Integrations Operator to your host.

   Tip: If you do not specify the CASE version, it will download the latest CASE.

   ```
   oc ibm-pak get \
   $CASE_NAME \
   --version $CASE_VERSION
   ```

By default, the root directory used by plug-in is `~/.ibm-pak`. This means that the preceding command will download the CASE under `~/.ibm-pak/data/cases/$CASE_NAME/$CASE_VERSION`. You can configure this root directory by setting the `IBMPAK_HOME` environment variable. Assuming `IBMPAK_HOME` is set, the preceding command will download the CASE under `$IBMPAK_HOME/.ibm-pak/data/cases/$CASE_NAME/$CASE_VERSION`.

The logs files will be available at `$IBMPAK_HOME/.ibm-pak/logs/oc-ibm_pak.log`.

Your host is now configured and you are ready to mirror your images.

Notes:

- Starting with v1.4.0, the plug-in creates a file, `component-set-config.yaml`, in the directory `~/.ibm-pak/data/cases/$CASE_NAME/$CASE_VERSION` to download the CASEs with `oc ibm-pak get`. This file captures all the CASEs that were downloaded, pinning down their exact versions during this particular download. You can use this file later to download the same CASEs with same versions in another environemnt. You can check in this file to your source code repository and recreate the same environment each time you use this to download the CASEs. Run the following command:

  ```
   oc ibm-pak get -c file:///home/user/ibm-pak/data/cases/$CASE_NAME/$CASE_VERSION/component-
  set-config.yaml
  ```

- – Note that the path after `file://` should be an absolute path.
- You can also edit this file defining the CASEs with pinned down versions which should include your product. The following is an example file, `my-csc.yaml`:

```
name: ibm-netcool-integrations
version: 3.4.1
description: Generated from CASE ibm-netcool-integrations version 3.4.1 on 2023-06-02
  20:13:17 PST
cases:
- name: ibm-netcool-integrations
  version: 3.4.1
  launch: true
```

Mirror images to your private container registry

The process of mirroring images takes the image from the internet to your host, then effectively copies that image to your private container registry. After you mirror your images, you can configure your cluster and complete air-gapped installation.

Complete the following steps to mirror your images from your host to your private container registry:

1. Generate mirror manifests
2. Authenticating the registry
3. Mirror images to final location
4. Configure the cluster
5. Install IBM Netcool Operations Insight Event Integrations Operator by way of Red Hat Open Shift Container Platform

1. Generate mirror manifests

Notes:

- If you want to install subsequent updates to your air-gapped environment, you must do a `CASE get` to get the image list when performing those updates. A registry namespace suffix can optionally be specified on the target registry to group mirrored images.
- Define the environment variable `$TARGET_REGISTRY` by running the following command:

  ```
  export TARGET_REGISTRY=<target-registry>
  ```

  The `<target-registry>` refers to the registry (hostname and port) where your images will be mirrored to and accessed by the oc cluster. For example setting TARGET_REGISTRY to `myregistry.com:5000/mynamespace` will create manifests such that images will be mirrored to the top-level namespace `mynamespace`.
- Run the following commands to generate mirror manifests to be used when mirroring from a bastion host (connected mirroring):

  ```
  oc ibm-pak generate mirror-manifests \
     $CASE_NAME \
     $TARGET_REGISTRY \
     --version $CASE_VERSION
  ```

  Example `~/.ibm-pak` directory structure for connected mirroring

  The `~/.ibm-pak` directory structure is built over time as you save CASEs and mirror. The following tree shows an example of the `~/.ibm-pak` directory structure for connected mirroring:

  ```
   tree ~/.ibm-pak
  /root/.ibm-pak
  ├── config
  │   └── config.yaml
  ├── data
  │   ├── cases
  │   │   └── ibm-netcool-integrations
  │   │       └── 3.4.1
  │   │           ├── caseDependencyMapping.csv
  │   │           ├── charts
  │   │           ├── component-set-config.yaml
  ```

```
│   │           ├── ibm-netcool-integrations-3.4.1-airgap-metadata.yaml
│   │           ├── ibm-netcool-integrations-3.4.1-charts.csv
│   │           ├── ibm-netcool-integrations-3.4.1-images.csv
│   │           ├── ibm-netcool-integrations-3.4.1.tgz
│   │           └── resourceIndexes
│   │               └── ibm-netcool-integrations-resourcesIndex.yaml
│   └── mirror
│       └── ibm-netcool-integrations
│           └── 3.4.1
│               ├── catalog-sources-linux-amd64.yaml
│               ├── image-content-source-policy.yaml
│               └── images-mapping.txt
└── logs
    └── oc-ibm_pak.log
```

Notes: A new directory `~/.ibm-pak/mirror` is created when you issue the

```
oc ibm-pak generate mirror-manifests
```

command. This directory holds the `image-content-source-policy.yaml`, `images-mapping.txt`, and `catalog-sources.yaml` files.

Tip: If you are using a Red Hat® Quay.io registry and need to mirror images to a specific organization in the registry, you can target that organization by specifying:

```
   export ORGANIZATION=<your-organization>
   oc ibm-pak generate mirror-manifests
   $CASE_NAME
   $TARGET_REGISTRY/$ORGANIZATION
   --version $CASE_VERSION
```

- Run the following commands to generate mirror manifests to be used when mirroring from a file system (disconnected mirroring):

```
oc ibm-pak generate mirror-manifests \
   $CASE_NAME \
   file://local \
   --final-registry $TARGET_REGISTRY
```

Example `~/.ibm-pak` directory structure for disconnected mirroring

- The following tree shows an example of the `~/.ibm-pak` directory structure for disconnected mirroring:

```
 tree ~/.ibm-pak
/root/.ibm-pak
├── config
│   └── config.yaml
├── data
│   ├── cases
│   │   └── ibm-netcool-integrations
│   │       └── 3.4.1
│   │           ├── caseDependencyMapping.csv
│   │           ├── charts
│   │           ├── component-set-config.yaml
│   │           ├── ibm-netcool-integrations-3.4.1-airgap-metadata.yaml
│   │           ├── ibm-netcool-integrations-3.4.1-charts.csv
│   │           ├── ibm-netcool-integrations-3.4.1-images.csv
│   │           ├── ibm-netcool-integrations-3.4.1.tgz
│   │           └── resourceIndexes
│   │               └── ibm-netcool-integrations-resourcesIndex.yaml
│   └── mirror
│       └── ibm-netcool-integrations
│           └── 3.4.1
│               ├── catalog-sources-linux-amd64.yaml
│               ├── image-content-source-policy.yaml
│               ├── images-mapping-from-filesystem.txt
│               ├── images-mapping-to-filesystem.txt
│               ├── images-mapping.txt
└── logs
    └── oc-ibm_pak.log
```

Note: A new directory `~/.ibm-pak/mirror` is created when you issue the oc `ibm-pak generate mirror-manifests` command. This directory holds the `image-`

`content-source-policy.yaml`, `images-mapping-to-filesystem.txt`, `images-mapping-from-filesystem.txt`, and `catalog-sources.yaml` files.

Tip: Some products support the ability to generate mirror manifests only for a subset of images using the `--filter` argument and image grouping. The `--filter` argument provides the ability to customize which images are mirrored during an air-gapped installation. As an example for this functionality `ibm-cloud-native-postgresql` CASE can be used, which contains groups that allow mirroring specific variant of `ibm-cloud-native-postgresql` (Standard or Enterprise). Use the `--filter` argument to target a variant of `ibm-cloud-native-postgresql` to mirror rather than the entire library. The filtering can be applied for groups and architectures. Consider the following command:

```
oc ibm-pak generate mirror-manifests \
   ibm-cloud-native-postgresql \
   file://local \
   --final-registry $TARGET_REGISTRY \
   --filter $GROUPS
```

The command was updated with a `--filter` argument. For example, for `$GROUPS` equal to `ibmEdbStandard` the mirror manifests will be generated only for the images associated with `ibm-cloud-native-postgresql` in its Standard variant. The resulting image group consists of images in the `ibm-cloud-native-postgresql` image group as well as any images that are not associated with any groups. This allows products to include common images as well as the ability to reduce the number of images that you need to mirror.

Note: You can use the following command to list all the images that will be mirrored and the publicly accessible registries from where those images will be pulled from:

```
oc ibm-pak describe $CASE_NAME --version $CASE_VERSION --list-mirror-images
```

Tip: Note down the `Registries found` section at the end of output from the preceding command. You will need to log in to those registries so that the images can be pulled and mirrored to your local target registry. See the next steps on authentication. `Top-level namespaces found` section shows the list of namespaces under which the images will be mirrored. These namespaces should be created manually in your registry root path if your registry doesn't allow automatic creation of the namespaces.

2. Authenticating the registry

Complete the following steps to authenticate your registries:

1. Store authentication credentials for all source Docker registries.

   Your product might require one or more authenticated registries. The following registries require authentication:

   - `cp.icr.io`
   - `registry.redhat.io`
   - `registry.access.redhat.com`

   You must run the following command to configure credentials for all target registries that require authentication. Run the command separately for each registry:

   Note: The `export REGISTRY_AUTH_FILE` command only needs to run once.

   ```
   export REGISTRY_AUTH_FILE=<path to the file which will store the auth credentials generated
   on podman login>
   podman login <TARGET_REGISTRY>
   ```

   Important: When you log in to `cp.icr.io`, you must specify the user as `cp` and the password which is your Entitlement key from the IBM Cloud Container Registry. For example:

   ```
   podman login cp.icr.io
   Username: cp
   ```

```
Password:
Login Succeeded!
```

For example, if you export REGISTRY_AUTH_FILE=~/.ibm-pak/auth.json, then after performing podman login, you can see that the file is populated with registry credentials.

If you use docker login, the authentication file is typically located at $HOME/.docker/config.json on Linux or %USERPROFILE%/.docker/config.json on Windows. After docker login you should export REGISTRY_AUTH_FILE to point to that location. For example in Linux you can issue the following command:

```
export REGISTRY_AUTH_FILE=$HOME/.docker/config.json
```

Table 6. Directory description

Directory	Description
~/.ibm-pak/config	Stores the default configuration of the plug-in and has information about the public GitHub URL from where the cases are downloaded.
~/.ibm-pak/data/cases	This directory stores the CASE files when they are downloaded by issuing the oc ibm-pak get command.
~/.ibm-pak/data/mirror	This directory stores the image-mapping files, ImageContentSourcePolicy manifest in image-content-source-policy.yaml and CatalogSource manifest in one or more catalog-sourcesXXX.yaml. The files images-mapping-to-filesystem.txt and images-mapping-from-filesystem.txt are input to the oc image mirror command, which copies the images to the file system and from the file system to the registry respectively.
~/.ibm-pak/data/logs	This directory contains the oc-ibm_pak.log file, which captures all the logs generated by the plug-in.

3. Mirror images to final location

Complete the steps in this section on your host that is connected to both the local Docker registry and the Red Hat® OpenShift® Container Platform cluster.

Mirror images to the final location.

- **For mirroring from a bastion host (connected mirroring):**

  Mirror images to the TARGET_REGISTRY:

```
oc image mirror \
  -f ~/.ibm-pak/data/mirror/$CASE_NAME/$CASE_VERSION/images-mapping.txt \
  --filter-by-os '.*'  \
  -a $REGISTRY_AUTH_FILE \
  --insecure  \
  --skip-multiple-scopes \
  --max-per-registry=1 \
  --continue-on-error=true
```

```
oc image mirror --help
```

The above command can be used to see all the options available on the mirror command. Note that we use `continue-on-error` to indicate that command should try to mirror as much as possible and continue on errors.

Note: Sometimes based on the number and size of images to be mirrored, the `oc image mirror` might take longer. If you are issuing the command on a remote machine it is recommended that you run the command in the background with a nohup so even if network connection to your remote machine is lost or you close the terminal the mirroring will continue. For example, the below command will start the mirroring process in background and write the log to `my-mirror-progress.txt`.

```
nohup oc image mirror \
-f ~/.ibm-pak/data/mirror/$CASE_NAME/$CASE_VERSION/images-mapping.txt \
-a $REGISTRY_AUTH_FILE \
--filter-by-os '.*' \
--insecure \
--skip-multiple-scopes \
--max-per-registry=1 \
--continue-on-error=true > my-mirror-progress.txt  2>&1 &
```

You can view the progress of the mirror by issuing the following command on the remote machine:

```
  tail -f my-mirror-progress.txt
```

- **For mirroring from a file system (disconnected mirroring):**

  Mirror images to your file system:

```
export IMAGE_PATH=<image-path>
oc image mirror \
  -f ~/.ibm-pak/data/mirror/$CASE_NAME/$CASE_VERSION/images-mapping-to-filesystem.txt \
  --filter-by-os '.*'  \
  -a $REGISTRY_AUTH_FILE \
  --insecure  \
  --skip-multiple-scopes \
  --max-per-registry=1 \
  --continue-on-error=true \
  --dir "$IMAGE_PATH"
```

  The `<image-path>` refers to the local path to store the images. For example, in the previous section if provided `file://local` as input during generate mirror-manifests, then the preceding command will create a subdirectory v2/local inside directory referred by `<image-path>` and copy the images under it.

The following command can be used to see all the options available on the mirror command. Note that `continue-on-error` is used to indicate that the command should try to mirror as much as possible and continue on errors.

```
  oc image mirror --help
```

Note: Sometimes based on the number and size of images to be mirrored, the `oc image mirror` might take longer. If you are issuing the command on a remote machine, it is recommended that you run the command in the background with nohup so that even if you lose network connection to your remote machine or you close the terminal, the mirroring will continue. For example, the following command will start the mirroring process in the background and write the log to `my-mirror-progress.txt`.

```
   export IMAGE_PATH=<image-path>
   nohup oc image mirror \
     -f ~/.ibm-pak/data/mirror/$CASE_NAME/$CASE_VERSION/images-mapping-to-filesystem.txt \
     --filter-by-os '.*' \
     -a $REGISTRY_AUTH_FILE \
     --insecure \
     --skip-multiple-scopes \
     --max-per-registry=1 \
     --continue-on-error=true \
     --dir "$IMAGE_PATH" > my-mirror-progress.txt  2>&1 &
```

You can view the progress of the mirror by issuing the following command on the remote machine:

```
tail -f my-mirror-progress.txt
```

1. **For disconnected mirroring only:** Continue to move the following items to your file system:
   - The `<image-path>` directory you specified in the previous step
   - The `auth` file referred by `$REGISTRY_AUTH_FILE`
   - `~/.ibm-pak/data/mirror/$CASE_NAME/$CASE_VERSION/images-mapping-from-filesystem.txt`
2. **For disconnected mirroring only:** Mirror images to the target registry from file system

   Complete the steps in this section on your file system to copy the images from the file system to the `$TARGET_REGISTRY`. Your file system must be connected to the target docker registry.

   Important: If you used the placeholder value of TARGET_REGISTRY as a parameter to `--final-registry` at the time of generating mirror manifests, then before running the following command, find and replace the placeholder value of TARGET_REGISTRY in the file, `images-mapping-from-filesystem.txt`, with the actual registry where you want to mirror the images. For example, if you want to mirror images to `myregistry.com/mynamespace` then replace TARGET_REGISTRY with `myregistry.com/mynamespace`.

   a. Run the following command to copy the images (referred in the `images-mapping-from-filesystem.txt` file) from the directory referred by `<image-path>` to the final target registry:

   ```
   export IMAGE_PATH=<image-path>
   oc image mirror \
     -f ~/.ibm-pak/data/mirror/$CASE_NAME/$CASE_VERSION/images-mapping-from-filesystem.txt \
     -a $REGISTRY_AUTH_FILE \
     --from-dir "$IMAGE_PATH" \
     --filter-by-os '.*' \
     --insecure \
     --skip-multiple-scopes \
     --max-per-registry=1 \
     --continue-on-error=true
   ```

4. Configure the cluster

1. Update the global image pull secret for your Red Hat OpenShift cluster. Follow the steps in Updating the global cluster pull secret.

   The documented steps in the link enable your cluster to have proper authentication credentials in place to pull images from your TARGET_REGISTRY as specified in the `image-content-source-policy.yaml` which you will apply to your cluster in the next step.
2. Create ImageContentSourcePolicy

   Important:

   - Before you run the command in this step, you must be logged into your OpenShift cluster. Using the `oc login` command, log in to the Red Hat OpenShift Container Platform cluster where your final location resides. You can identify your specific oc login by clicking the user drop-down menu in the Red Hat OpenShift Container Platform console, then clicking **Copy Login Command**.
     – If you used the placeholder value of TARGET_REGISTRY as a parameter to `--final-registry` at the time of generating mirror manifests, then before running the following command, find and replace the placeholder value of TARGET_REGISTRY in file, `~/.ibm-pak/data/mirror/$CASE_NAME/$CASE_VERSION/image-content-source-policy.yaml` with the actual registry where you want to mirror the images. For example, replace TARGET_REGISTRY with `myregistry.com/mynamespace`.

   Run the following command to create ImageContentSourcePolicy:

   ```
      oc apply -f  ~/.ibm-pak/data/mirror/$CASE_NAME/$CASE_VERSION/image-content-source-
   policy.yaml
   ```

If you are using Red Hat OpenShift Container Platform version 4.7 or earlier, this step might cause your cluster nodes to drain and restart sequentially to apply the configuration changes.

3. Verify that the ImageContentSourcePolicy resource is created.

```
oc get imageContentSourcePolicy
```

4. Verify your cluster node status and wait for all the nodes to be restarted before proceeding.

```
oc get MachineConfigPool
```

```
$ oc get MachineConfigPool -w
NAME     CONFIG                                             UPDATED   UPDATING   DEGRADED
MACHINECOUNT   READYMACHINECOUNT   UPDATEDMACHINECOUNT   DEGRADEDMACHINECOUNT   AGE
master   rendered-master-53bda7041038b8007b038c08014626dc   True      False      False
3              3                   3                     0                      10d
worker   rendered-worker-b54afa4063414a9038958c766e8109f7   True      False      False
3              3                   3                     0                      10d
```

After the `ImageContentsourcePolicy` and global image pull secret are applied, the configuration of your nodes will be updated sequentially. Wait until all `MachineConfigPools` are in the `UPDATED=True` status before proceeding.

5. Create a new project for the CASE commands by running the following commands:

   Note: You must be logged into a cluster before performing the following steps.

```
export NAMESPACE=<YOUR_NAMESPACE>
```

```
oc new-project $NAMESPACE
```

6. **Optional:** If you use an insecure registry, you must add the target registry to the cluster insecureRegistries list.

```
oc patch image.config.openshift.io/cluster --type=merge \
-p '{"spec":{"registrySources":{"insecureRegistries":["'${TARGET_REGISTRY}'"]}}}'
```

7. Verify your cluster node status and wait for all the nodes to be restarted before proceeding.

```
oc get MachineConfigPool -w
```

After the `ImageContentsourcePolicy` and global image pull secret are applied, the configuration of your nodes will be updated sequentially. Wait until all `MachineConfigPools` are updated.

5. Install IBM Netcool Operations Insight Event Integrations Operator by way of Red Hat Open Shift Container Platform

Now that your images are mirrored to your air-gapped environment, you can deploy your IBM Cloud® Paks to that environment. When you mirrored your environment, you created a parallel offline version of everything that you needed to install an operator into Red Hat® OpenShift® Container Platform.

5.1 Create the catalog source and install the IBM Netcool Operations Insight Event Integrations Operator

Important: Before you run any of the `oc ibm-pak launch \` command, you must be logged into your cluster. Using the`oc login` command, log in to the Red Hat OpenShift Container Platform cluster where your final location resides. You can identify your specific oc login by clicking the user drop-down menu in the Red Hat OpenShift Container Platform console, then clicking **Copy Login Command**.

1. Set the namespace to install the IBM Netcool Operations Insight Event Integrations Operator catalog:

```
export NAMESPACE=<YOUR_NAMESPACE>
```

2. Set the environment variable of the `--inventory` parameter:

```
export CASE_INVENTORY_SETUP=netcoolIntegrationsOperatorSetup
```

3. Create and configure a catalog source.

   The recommended way to install the catalog is to run the following command:

```
oc apply -f ~/.ibm-pak/data/mirror/$CASE_NAME/$CASE_VERSION/catalog-sources-linux-amd64.yaml
```

   The following command can also be used to install the catalog:

```
   oc ibm-pak launch \
   $CASE_NAME \
     --version $CASE_VERSION \
     --action install-catalog \
     --inventory $CASE_INVENTORY_SETUP \
     --namespace $NAMESPACE \
     --args "--registry $TARGET_REGISTRY --recursive \
     --inputDir ~/.ibm-pak/data/cases/$CASE_NAME/$CASE_VERSION"
```

   .

4. Verify that the `CatalogSource` for your IBM Netcool Operations Insight Event Integrations Operator is created.

```
oc get pods -n openshift-marketplace
oc get catalogsource -n openshift-marketplace
```

5. Install your IBM Netcool Operations Insight Event Integrations Operator.

   Note: You must have cluster admin access to run the `install-operator` command. However, you do not need cluster admin access for mirroring.

```
oc ibm-pak launch \
   $CASE_NAME \
   --version $CASE_VERSION \
   --inventory $CASE_INVENTORY_SETUP \
   --action install-operator \
   --namespace $NAMESPACE
```

6. Using the `oc login` command, log in to the Red Hat® OpenShift® Container Platform cluster where your final location resides. You can identify your specific `oc login` by clicking the user drop-down menu in the Red Hat OpenShift Container Platform console, then clicking **Copy Login Command**.
7. Verify that your IBM Netcool Operations Insight Event Integrations Operator is installed:

```
oc get pod -n $NAMESPACE
```

   It might take up to 15 minutes for all the pods to show the `Running` status.

Set up a repeatable mirroring process

Once you complete a CASE save, you can mirror the CASE as many times as you want to. This approach allows you to mirror a specific version of the IBM Netcool Operations Insight Event Integrations Operator into development, test, and production stages using a private container registry.

Follow the steps in this section if you want to save the CASE to multiple registries (per environment) once and be able to run the CASE in the future without repeating the CASE save process.

1. Run the following command to save the CASE to ~/.ibm-pak/data/cases/$CASE_NAME/ $CASE_VERSION which can be used as an input during the mirror manifest generation:

```
oc ibm-pak get \
$CASE_NAME \
--version $CASE_VERSION
```

2. Run the `oc ibm-pak generate mirror-manifests` command to generate the `image-mapping.txt`:

   ```
   oc ibm-pak generate mirror-manifests \
   $CASE_NAME \
   $TARGET_REGISTRY \
   --version $CASE_VERSION
   ```

   Note: If you are using a Red Hat® Quay.io registry and need to mirror images to a specific organization in the registry, you can target that organization by specifying:

   ```
   oc ibm-pak generate mirror-manifests \
      $CASE_NAME \
      $TARGET_REGISTRY \
      --version $CASE_VERSION
   ```

   Then add the `image-mapping.txt` to the `oc image mirror` command:

   ```
   oc image mirror \
     -f ~/.ibm-pak/data/mirror/$CASE_NAME/$CASE_VERSION/images-mapping.txt \
     --filter-by-os '.*'  \
     -a $REGISTRY_AUTH_FILE \
     --insecure  \
     --skip-multiple-scopes \
     --max-per-registry=1 \
     --continue-on-error=true
   ```

If you want to make this repeatable across environments, you can reuse the same saved CASE cache (`~/.ibm-pak/$CASE_NAME/$CASE_VERSION`) instead of executing a CASE save again in other environments. You do not have to worry about updated versions of dependencies being brought into the saved cache.

Verifying the installation

After installing the operator and operands. you can verifying the installation.

To verify the installation, use the following steps:

1. Check the status of each custom resource.

   ```
    oc get prometheusprobe,kafkaprobe,pulsarprobe,jdbcgateway,pulsargateway  -o custom-
   columns="NAMESPACE:metadata.namespace,NAME:metadata.name,PHASE:status.phase"
   ```

2. Verify that the PHASE status of each instance is `OK`.
3. Check status of the pods.

   ```
   oc get pods
   ```

4. Verify that all pods status is `Running`.

Phase status

The **`Phase status`** property provides the current phase of the probe or gateway instances. This property can take one the following values:

`Initializing`: The operator is processing the operand custom resource and initializing the required Kubernetes resources.

`Creating`: The operator is creating the Kubernetes resources.

`Reconciling`: The operator is reconciling the operand and may perform updates.

`OK`: There are no errors. The instance was created successfully and is running.

`Error`: There was a `ReconcileError`.

Operator Upgrade and Rollback process

You can upgrade using the Operator Lifecycle Manager (OLM) UI, or with the command line interface.

Upgrading with the OLM UI

Use these instructions to upgrade an existing IBM Netcool Operations Insight Event Integrations Operator using the Red Hat® OpenShift® Operator Lifecycle Manager (OLM) user interface (UI).

Note: Before upgrading, review the changes in the latest operator version in the "What's new" on page 1 page.

Upgrade the Catalog Source

1. From the Red Hat OpenShift OLM UI, navigate to **Administration > Cluster Settings**, and then select the **OperatorHub** configuration resource under the Global Configurations tab.
2. Under the **Sources** tab, click the existing IBM Netcool Operations Insight Event Integrations catalog source.
3. Edit the catalog source YAML and upgrade image (`spec.image`) with the IBM Netcool Operations Insight Event Integrations catalog source image with image tag, for example:

   ```
   icr.io/cpopen/ibm-netcool-integrations-operator-catalog:<latest version>
   ```

   or with the image with digest, where `<latest version>` is the latest catalog source image tag. Click the **Save** button.

Upgrade the IBM Netcool Operations Insight Event Integrations operator

To upgrade to the latest operator version, the operator subscription must be updated to the latest subscription channel. You may skip this step if your operator is already subscribed to the latest update channel available and the Installation mode is specified as `Automatic`.

1. Navigate to **Operators > Installed Operators**, select **Project**, search for the IBM Netcool Operations Insight Event Integrations operator and select the operator.
2. Go to the **Subscription** tab, under **Subscription Details**, change the **Channel** to the latest subscription channel and click **Save**. The subscription channel follows the `vX.Y` format where X is the major version and Y is the minor version.
3. Navigate to **Operators > Installed Operators** and view the IBM Netcool Operations Insight Event Integrations operator. It takes several minutes for the operator to upgrade. Ensure that the operator status changes from `Upgrading` to `Succeeded`. If the **Status** shows `UpgradePending`, it is likely that the operator was installed with `Manual` approval strategy. Click on the operator name and check if the upgrade is pending an approval. To approve the upgrade, under subscription details, click the **requires approval** button > **Preview Install Plan** > click **Approve**.

Upgrade the Probe or Gateway instances

1. Navigate to **Operators > Installed Operators**, select **Project**, and search for and select the IBM Netcool Operations Insight Event Integrations operator.
2. Go to the **All instances** tab and select the probe or gateway instance to update.
3. Edit the probe or gateway custom resource YAML and update the version specification `spec.version`. It is recommended that you take a copy of the instance YAML before changing it, in case you later decide to rollback. For more information about configurable properties for each integration, see:
   - Deploying the Probe for Prometheus Integration
   - Deploying the Gateway for Kafka Integration
   - Deploying the JDBC Gateway Integration
   - Deploying the Gateway for Pulsar Integration
   - Deploying the Probe for Pulsar Integration

- “Deploying the Generic Webhook Probe” on page 130
- “Deploying the Probe for Kafka Integration” on page 80

4. Click the **Save** button.
5. Navigate to **Operators > Installed Operators**, and select **Project**. Search for and select the IBM Netcool Operations Insight Event Integrations operator.
6. Under the **All Instances** tab, view the status of each of the updates on the installation. When the instance's status shows **OK**, then the upgrade is successful. Optionally, verify that all pods are running to ensure that the upgrade has completed successfully.

Upgrading with the CLI

Use these instructions to upgrade an existing IBM Netcool Operations Insight Event Integrations Operator using the command line interface (CLI).

Note: Before upgrading, review the changes in the latest operator version in the “What's new” on page 1 page and note the latest operator version.

Upgrade the Catalog Source

Note: These steps are only required if the catalog image is using a specific version tag, for example: `v3.6.0-amd64`. If the latest tag is used, these steps can be skipped.

1. Login to the cluster using the `oc` client.
2. In the target namespace, search for an existing `CatalogSource` resource for the IBM Netcool Operations Insight Event Integrations operator:

   ```
   kubectl get catalogsource --namespace $namespace
   ```

   If it is not in the target namespace, run the same command in the `openshift-marketplace` namespace.
3. Update the catalog source resource and upgrade image (`spec.image`) with the IBM Netcool Operations Insight Event Integrations catalog source image with image tag, for example:

   ```
   icr.io/cpopen/ibm-netcool-integrations-operator-catalog:<latest version>
   ```

   or with the image with digest, where `<latest version>` is the latest catalog source image tag. Click the **Save** button.
4. Give a few minutes for the operator catalog to update and verify that the operator pod is restarted and it is in `Running` state.

Upgrade the IBM Netcool Operations Insight Event Integrations operator

To upgrade to the latest operator version, the operator subscription must be updated to the latest subscription channel. You may skip this step if your operator is already subscribed to the latest update channel available and the Installation mode is specified as `Automatic`.

1. Get the existing `Subscription` resource.

   ```
   kubectl get subscription --namespace $namespace
   kubectl describe subscription <subscription name> --namespace $namespace
   ```

   Replace `<subscription name>` with the name of the existing Subscription resource.
2. Edit the Subscription resource from the previous step and change the `spec.channel` to the latest available subscription channel. The subscription channel follows the `vX.Y` format where X is the major version and Y is the minor version of the operator.
3. Describe the new `InstallPlan` resource for the new operator version:

   ```
   kubectl get installplan --namespace $namespace
   kubectl describe <install-plan resource>
   ```

Replace <install-plan resource> with the name of the resource.

4. If the Approved field shows RequiresApproval or false, edit the InstallPlan resource and change the value of spec.approved to true.
5. Run the following command to get the .status.phase field of the InstallPlan resource and ensure that the value of changes to Complete:

   ```
   kubectl get installplan <install-plan resource> --namespace $namespace -o
   jsonpath='{.status.phase}'
   ```

Upgrade the Probe or Gateway instances

1. Run the command below to obtain the existing instance resources in the target namespace:

   ```
   kubectl get
   prometheusprobe,pulsarprobe,jdbcgateway,pulsargateway,kafkaprobe,webhookprobe --namespace
   $namespace
   ```

2. Edit the probe or gateway instance and update its version specification (spec.version) using the kubectl edit <resource type> <resource name> command. It is recommended that you take a copy of the instance YAML before changing it, in case you later decide to rollback. For more information about configurable properties for each integration, see:
   - Deploying the Probe for Prometheus Integration
   - Deploying the Gateway for Kafka Integration
   - Deploying the JDBC Gateway Integration
   - Deploying the Gateway for Pulsar Integration
   - Deploying the Probe for Pulsar Integration
   - "Deploying the Generic Webhook Probe" on page 130
   - "Deploying the Probe for Kafka Integration" on page 80
3. Save the custom resource YAML.
4. Verify that the instance status shows **OK** that indicates the that upgrade was successful. Optionally, verify that all pods are running to ensure that the upgrade has completed successfully.

Rolling back process

Use these instructions to roll back an existing IBM Netcool Operations Insight Event Integrations Operator using the command line interface (CLI).

Before rolling back an operator, all instances that are running must be rolled back to the supported version of the previous operator.

Rolling back probe or gateway instances

1. Get the instance name:

   ```
   kubectl get prometheusprobe,kafkaprobe --namespace $namespace
   ```

2. For each probe or gateway instance that was created with the current operator, revert the spec.version property to the previous version. Refer to the configurable parameter page of each integration kind for the supported versions.
3. Give a moment for the operator to process the update.

Rolling back the operator

To rollback the operator, you will need to uninstall the operator, re-install the operator version from the previous subscription channel.

Note You should only roll back the operator if there is a critical error in the operator and it is unable to recover after restarting the pod. Errors in the operator could be caused by an invalid configuration in the instance custom resource or version mismatch. Review your custom resource specification to investigate the issue.

Caution Uninstalling the operator while there are probe or gateway instances running may cause issues on the managed resources. It is recommended to uninstall or delete all instances before uninstalling the operator.

1. From the Red Hat OpenShift OLM UI, navigate to **Operators > Installed Operators**, select **Project**, and search for and select the IBM Netcool Operations Insight Event Integrations Operator. Under the **All Instances** tab, delete the instances. To delete instances via the command line, refer to the "Deleting an integration instance" on page 49 page for more info.
2. Navigate to **Operators > Installed Operators**, select **Project**, search for and select the IBM Netcool Operations Insight Event Integrations Operator and uninstall the operator.
3. Select the menu navigation **OperatorHub** and search for the **IBM Netcool Operations Insight Event Integrations** operator. Click the **Install** button.
4. Select the previous update channel and select a namespace to install the operator. Do not use namespaces that are kubernetes or openshift owned like kube-system or default. If you do not already have a project, create a project under the navigation menu **Home -> Projects**.
5. Click the **Install** button.

Installing integration resources using the CLI

This method installs a probe or gateway using the CASE launch scripts. The launch script is executed using the `cloudctl` tool and will perform the necessary prerequisite checks for each inventory action that is performed.

Note: The launch scripts are provided the CASE from version 1.1.0.

Prerequisites:

- The IBM Netcool Operations Insight Event Integration Operator is installed in the target namespace.
- CASE archive downloaded and unpacked into your local file system.

1. Extract the custom resource template YAML files from the CASE archive.

```
tar -xvf $CASEPATH ibm-netcool-integrations/inventory/netcoolIntegrationsOperator/files/
deploy/crs
```

2. Configure the integrations by configuring the probe or gateway custom resource template YAML file provided in the CASE archive in the `ibm-netcool-integrations/inventory/netcoolIntegrationsOperator/files/deploy/crs` directory.
3. Run the `apply-custom-resources` action in the launch script and set the updated custom resource YAML file path using the `--crFile` for each custom resource to install. More than one instance of probe and gateway can be deployed in the same namespace. **Note:** The CR instance name must be unique to avoid conflicts.

```
cloudctl case launch --case $CASEPATH \
 --namespace $NAMESPACE     \
 --inventory netcoolIntegrationsOperator   \
 --action apply-custom-resources  \
 --args "--crFile $CRFILE" \
 --tolerance 1
```

Configuring the global operands resources ConfigMap

For each operand, the pod resources requests and limits can be configured in its respective custom resource YAML file and it is applied to the specific instance only.

You can opt to create a global operands resources `ConfigMap` to override all resources requests and limits of related operands. The global operands resources `ConfigMap` must have the labels of `app.kubernetes.io/name: netcool-integrations-operator-global-operands-`

resources-config and app.kubernetes.io/component: netcool-integrations-operator-global-operands-resources-config to allow the operator to retrieve the data from the ConfigMap. The ConfigMap defines resources requests and limits per operand kind. All the same kind of operands will be updated according to the ConfigMap.

Note:

- If this ConfigMap is created, the resources limits and requests in the respective custom resource specification is ignored and the resource limits and requests set in this ConfigMap is applied to each operand instance.
- When the operator is installed in single namespace mode, only one global operands resources ConfigMap is allowed per namespace. If more than one global operands resources ConfigMap is created in the same namespace, the operator will deploy operands according to resources limits and requests defined in the respective custom resource specification.
- When the operator is installed in all namespaces mode, only one global operands resources ConfigMap is allowed on the cluster. If more than one global operands resources ConfigMap is created on the cluster, the operator will deploy operands according to resources limits and requests defined in the respective custom resource specification.

The following example shows a ConfigMap named netcool-integrations-operator-global-operands-resources-config. This ConfigMap contains resources limits and requests of all supported operands.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: netcool-integrations-operator-global-operands-resources-config
  labels:
    app.kubernetes.io/name: netcool-integrations-operator-global-operands-resources-config
    app.kubernetes.io/instance: netcool-integrations-operator-global-operands-resources-config
    app.kubernetes.io/component: netcool-integrations-operator-global-operands-resources-config
data:
  config: |
    cemprobes:
      resources:
        limits:
          cpu: 200m
          memory: 512Mi
          ephemeral-storage: 300Mi
        requests:
          cpu: 100m
          memory: 64Mi
          ephemeral-storage: 100Mi
    kafkaprobes:
      resources:
        limits:
          cpu: 200m
          memory: 512Mi
          ephemeral-storage: 300Mi
        requests:
          cpu: 100m
          memory: 64Mi
          ephemeral-storage: 100Mi
    prometheusprobes:
      resources:
        limits:
          cpu: 200m
          memory: 512Mi
          ephemeral-storage: 300Mi
        requests:
          cpu: 100m
          memory: 64Mi
          ephemeral-storage: 100Mi
    pulsarprobes:
      resources:
        limits:
          cpu: 200m
          memory: 512Mi
          ephemeral-storage: 300Mi
        requests:
          cpu: 100m
          memory: 64Mi
          ephemeral-storage: 100Mi
    webhookprobes:
      resources:
        limits:
          cpu: 200m
          memory: 512Mi
          ephemeral-storage: 300Mi
        requests:
          cpu: 100m
          memory: 64Mi
          ephemeral-storage: 100Mi
    cemgateways:
      resources:
        limits:
          cpu: 200m
          memory: 512Mi
          ephemeral-storage: 300Mi
        requests:
          cpu: 100m
          memory: 64Mi
          ephemeral-storage: 100Mi
    jdbcgateways:
      resources:
        limits:
          cpu: 200m
          memory: 512Mi
          ephemeral-storage: 300Mi
        requests:
          cpu: 100m
          memory: 64Mi
          ephemeral-storage: 100Mi
    pulsargateways:
      resources:
        limits:
          cpu: 200m
          memory: 512Mi
          ephemeral-storage: 300Mi
        requests:
          cpu: 100m
          memory: 64Mi
          ephemeral-storage: 100Mi
    kafkagateways:
      resources:
        limits:
          cpu: 500m
          memory: 512Mi
          ephemeral-storage: 500Mi
        requests:
          cpu: 250m
          memory: 256Mi
          ephemeral-storage: 100Mi
```

You can update this `ConfigMap` to exclude the operand kinds that you do not want to be managed by the global `ConfigMap` either by commenting out or removing the related lines. Then save the `ConfigMap` into a YAML file named `netcool-integrations-operator-global-operands-resources-config.yaml`.

The following command shows how to apply the `ConfigMap`.

```
oc apply -f netcool-integrations-operator-global-operands-resources-config.yaml
```

Providing custom probe rules in a ConfigMap

The operator creates a ConfigMap containing a default probe rules for every probe instance installed. If you want to customize the probe rules, you can provide a custom probe rules file in a ConfigMap.

You can opt to extract the default probe rules files to use as a template or create the ConfigMap with your own probe rules file. This ConfigMap must use `message_bus.rules` as the data key which contains the probe rules. Next is to reconfigure the probe custom resource and set `spec.helmValues.probe.rulesConfigmap` to the name of your ConfigMap.

1. Determine the probe instance name and get the ConfigMap name contains the probe rules file. The ConfigMap with `rules` suffix contains the probe rules file. The following command uses

webhookprobe as probe instance name to query the ConfigMap using the app.kubernetes.io/instance label. The --namespace option can also be specified if required.

```
$ oc get configmap -l app.kubernetes.io/instance=webhookprobe | grep 'rules'
webhookprobe-mb-webhook-rules            1      9d
```

Note the rules file name from the Data field. You need to extract the files that you want to customise. In this example, we will customise the message_bus.rules.

2. Describe the default rules ConfigMap to determine the data key to extract.

```
$ oc describe configmap webhookprobe-mb-webhook-rules
Name:         webhookprobe-mb-webhook-rules
Namespace:    noi-integrations
Labels:       app.kubernetes.io/component=mb-wh
            app.kubernetes.io/instance=webhookprobe
            app.kubernetes.io/managed-by=Tiller
            app.kubernetes.io/name=mb-webhook
            hdm.ibm.com/chartversion=3.2.0
            hdm.ibm.com/lastreconciled=2023-03-03-05-01-09-13708742-0000-UTC-m-29-026518990
            hdm.ibm.com/resourceowner=
            helm.sh/chart=ibm-netcool-probe-messagebus-webhook-prod
            release=webhookprobe
Annotations:  hdm.ibm.com/lastknownstate: 34bfcc5af412dc8e3d3a70c6801ee790

Data
====
message_bus.rules:
----
<message_bus.rules file content omitted>

Events:  <none>
```

3. To extract the default probe rules file which is message_bus.rules in the example above, run oc extract configmap/webhookprobe-mb-webhook-rules --keys=message_bus.rules --to=/tmp. This will extract the file from the ConfigMap into your local directory /tmp.
4. Customise the /tmp/message_bus.rules and save the file.
5. Run oc create configmap <custom_probe_rules_configmap_name> --from-file=message_bus.rules=/tmp/message_bus.rules to create a ConfigMap from the /tmp/message_bus.rules file.
6. Edit your probe custom resource and set spec.helmValues.probe.rulesConfigmap to the custom probe rules ConfigMap name created in the previous step.
7. The probe pod should restart to load the new rules files. To manually restart the probe pods, you can scale down and then scale up the probe deployment resource using the command below.

```
oc scale deployment webhookprobe-mb-webhook-server --replicas=0
oc scale deployment webhookprobe-mb-webhook-server --replicas=1
```

Viewing the health of an operator

This section describes how to view the health of an operator.

To display the health status of the operator, run the following commands:

```
kubectl describe deployment netcool-integrations-operator --namespace <NAMESPACE>
kubectl describe pod -l app.kubernetes.io/instance==netcool-integrations-operator --namespace
<NAMESPACE>
```

To view the Operator pod logs to check if there are any errors, run the following command:

```
kubectl logs -l app.kubernetes.io/instance==netcool-integrations-operator --namespace
<NAMESPACE>
```

Enabling probe self-monitoring

You can configure probes to generate ProbeWatch Heartbeat events as a self-monitoring mechanism to help monitor performance, diagnose performance problems, and highlight possible performance bottlenecks before they begin to affect the system.

To enable default probe self monitoring, you must configure `ProbeWatchHeartbeatInterval` to a positive number in custom resource YAML file to generate the probe heartbeart messages. `ProbeWatchHeartbeatInterval` is the interval in seconds at which probe heartbeart messages are generated. If `ProbeWatchHeartbeatInterval` is set to `0` or negative number, then no probe heartbeart messages will be generated. The default is `0`. This feature is supported by Kafka, Pulsar and Webhook probes.

See the configurable parameters table to toggle additional self-monitoring functions.

The example below generates probe heartbeat messages at every `60` seconds.

```
# Additional probe properties.
# Each property must not contain space.
# Do not put password and username related properties in the configs field below.
configs: >-
  ProbeWatchHeartbeatInterval:60
```

You can set `spec.helmValues.probe.selfMonitoring.populateMasterProbeStat` to `true` in custom resource YAML file to store OplStats probe metric in `master.probestats` table in ObjectServer.

You must use one of the options below to add a set of tables and triggers to ObjectServer when it is NOI 1.6.7 or earlier on OCP. You can skip the steps below when it is NOI 1.6.8 or later on OCP.

Running probestats.sql on ncoprimary and ncobackup pods

1. Run `oc get secret <noi_release_name>-omni-secret -o go-template --template="{{.data.OMNIBUS_ROOT_PASSWORD|base64decode}}"` to retrieve OMNIbus root passowrd.
2. Run `oc exec -it <noi_release_name>-ncoprimary-0 /bin/bash` to access `ncoprimary` pod.
3. Run `$NCHOME/omnibus/bin/nco_sql -user root -password password -server servername < $NCHOME/omnibus/extensions/roi/probestats.sql` to create the required triggers and tables.
4. Repeat steps 2 and 3 to update the `ncobackup` pod.

In the command above, `password` is the OMNIbus root password, `servername` is the name of the ObjectServer. The tables and triggers will be removed when restarting the `ncoprimary` and `ncobackup` pods.

Adding the SQL commands and triggers into ObjectServer ConfigMap

1. Run `oc rsync <noi_release-name>-ncoprimary-0:/opt/IBM/tivoli/netcool/omnibus/extensions/roi/probestats.sql .` to copy `probestats.sql` from `ncoprimary` pod to local directory.
2. Refer to Primary Netcool/OMNIbus ObjectServer configmap to add the contents of `probestats.sql` into `agg-p-sql-extensions` field in ObjectServer ConfigMap.
3. Refer to Backup Netcool/OMNIbus ObjectServer configmap to add the contents of `probestats.sql` into `agg-b-sql-extensions` field in ObjectServer ConfigMap.

The tables and triggers will be reloaded when restarting the `ncoprimary` and `ncobackup` pods.

Providing Pre-created Probe Rules ConfigMap to include probewatch.include File

You must pre-create ConfigMap that contains `message_bus.rules` and `probewatch.include` keys. `message_bus.rules` key contains the main probe rules file contents. `probewatch.include` key contains ProbeWatch include file contents.

The example below embeds `probewatch.include` file in main probe rules by using an `include` statement. The path in the `include` statement must point to `$PROBE_RULES_HOME/probewatch.include`.

```
if( match( @Manager, "ProbeWatch" ) )
{
    include "$PROBE_RULES_HOME/probewatch.include"
}
```

Monitoring event statistics

Event load profiling is enabled by default and can be disabled by setting `EventLoadProfiling` configuration parameter to `'false'` in custom resource configuration. The Event Statistics function enables several event statistics probe rules functions that can be used to log event statistics.

The following table describes the rules functions that are available when the Event Statistics function is enabled. This feature is supported by the Kafka, Pulsar and Webhook probes.

Function name	Description

`get_endpoint_load (endpoint_name)`	This function reports the total event count at the endpoint. Invalid endpoint or disabled event statistics will return a negative value.
	Example:
	`$ns_endpoint = "PRE_RULE_EVENTS"`
	`$ns_endpoint_load =`
	`get_endpoint_load($ns_endpoint)`
	`log(DEBUG, $ns_endpoint + " endpoint load:`
	`[" + $ns_endpoint_load + "]")`

`get_endpoint_event_count (endpoint_name)`	This function reports the total event count at the endpoint. Invalid endpoint or disabled event statistics will return a negative value.
	Example:
	`$ns_event_count =`
	`get_endpoint_event_count("PRE_RULE_EVENTS")`
	`log(DEBUG, $ns_endpoint + " endpoint event`
	`count: [" + $ns_event_count + "]")`

`get_endpoint_names()`	This function returns the names of the endpoint counter in a semicolon-delimited list. At least one endpoint name will be returned, that is PRE_RULE_EVENTS.
	Example:
	`$obtained_endpoint_name =`
	`get_endpoint_names()`
	`log(DEBUG, " endpoint name: [" +`
	`$obtained_endpoint_name + "]")`

Function name	Description

`get_endpoint_annotation (endpoint_name)`	This function returns the annotation describing the endpoint. For example, it may include a description of the type of events being measured.
	Example:
	`$obtained_endpoint_annotation = get_endpoint_annotation($obtained_endpoint_name)`
	`log(DEBUG, " endpoint annotation: [" + $obtained_endpoint_annotation + "]")`

Pulling images from IBM Entitled Registry

To pull images from IBM Entitled Registry, use the following steps:

1. Log in to MyIBM Container Software Library with the IBMid and password that are associated with the entitled software.
2. In the **Entitlement keys** section, click **Copy key** to copy the entitlement key to the clipboard.
3. Save the APIkey.
4. Set the entitled registry information. Run export commands that set ENTITLED_REGISTRY to `cp.icr.io`, set ENTITLED_REGISTRY_USER to cp, and set ENTITLED_REGISTRY_KEY to the entitlement key that you got from the previous step.

   ```
   export ENTITLED_REGISTRY=cp.icr.io
   export ENTITLED_REGISTRY_USER=cp
   export ENTITLED_REGISTRY_KEY=<apikey>
   ```

5. Log in to the entitled registry with the following docker login command:

   ```
   docker login "$ENTITLED_REGISTRY" -u "$ENTITLED_REGISTRY_USER" -p "$ENTITLED_REGISTRY_KEY"
   ```

6. You can now pull images to your local file system.

Listing images in IBM Entitled Registry

Prerequisite: IBM Cloud CLI tool.

To install the `ibmcloud` CLI tool, see: https://cloud.ibm.com/docs/cli?topic=cli-install-ibmcloud-cli

To list images on the IBM Entitled Registry, use the following steps:

1. Install the Container Registry plug-in:

   ```
   ibmcloud plugin install container-registry -r 'IBM Cloud'
   ```

2. Log in to your IBM Cloud account:

   ```
   ibmcloud login -a https://cloud.ibm.com
   ```

3. Set the region as global:

   ```
   ibmcloud cr region-set global
   ```

4. List the available images by using the following command. This lists all Netcool Operations Insight Integrations images in the `noi-int` namespace:

   ```
   ibmcloud cr image-list --include-ibm | grep -I noi-int
   ```

Configuring custom resources

This section describes how to configure a custom resource (probe or gateway) instance.

To install a probe or gateway instance, you need to deploy a custom resource YAML in the namespace which is being watched by the operator. See "Installing the IBM Netcool Operations Insight Event Integrations Operator" on page 21.

For details about how to update the YAML for your environment, refer to the deploying page of the custom resource. See Integrations for AIOps.

Updating an integration instance

This section describes how to update an integration instance.

To update a probe or gateway instance, you can edit the specific custom resource deployed in the namespace. For example, to update a `PrometheusProbe` instance, use the following steps:

1. Get the instance resource YAML:

   ```
   kubectl get prometheusprobe example-prometheusprobe --namespace <NAMESPACE> -o yaml > /tmp/
   example-prometheusprobe-cr.yaml
   ```

2. Update the `/tmp/example-prometheusprobe-cr.yaml` configuration YAML and then deploy the updated YAML file:

   ```
   kubectl apply -f /tmp/example-prometheusprobe-cr.yaml
   ```

Some probe and gateway configuration are exposed in `Configmap` resources such as probe rules files, probe and gateway properties file, and gateway table mapping and replication definitions. These configurations can be changed by updating the `Configmap` resource directly. The operator should then perform the necessary updates to the pods. If required, you can scale down the number of replicas to zero and then scale up to the desired replica count to ensure that the probe and gateway pods reload the updated configurations from the `Configmap`.

Deleting an integration instance

This section describes how to delete an integration instance.

You can delete an integration instance by deleting the custom resource in the namespace. For example, to delete a `PrometheusProbe` resource, run the following command:

```
kubectl delete prometheusprobe example-prometheusprobe  --namespace <NAMESPACE>
```

Uninstalling the operator

You can uninstall the operator using the Operator Lifecycle Manager (OLM) UI, or with the command line interface.

Before deleting the operator, ensure that you have deleted all instances by deleting the custom resources in the namespace. This is to ensure that all resources created by the existing instances are deleted properly. You may need to allow some time for the operator to delete the resources before deleting the operator deployment.

Using the OLM UI

This section describes how to delete the operator using the OLM UI.

1. Go to **Operator > Installed Operators**.
2. Select the project where you installed Netcool Operations Insight. Click **IBM Netcool Operations Insight Event Integrations Operator > All Instances** and delete the probe and gateway instances.

3. Delete the IBM Netcool Operations Insight Event Integrations Operator. Go to **Operator > Installed Operators**. Select the options menu for the IBM Netcool Operations Insight Event Integrations operator entry, and select **Uninstall Operator**.
4. Remove the catalog source entry. Go to **Administration > Cluster Settings > Global Configuration > OperatorHub > Sources**. Select the catalog source entry serving the operator and select **Delete CatalogSource**.
5. Delete the Custom Resource Definitions (CRDs). Go to **Administration > Custom Resource Definitions**, search for **integrations.noi** to select the CRDs under this group that were created by the operator installation and delete all the CRDs.

Using the CLI with CASE

This section describes how to uninstall the operator using the CASE launch script for offline airgap deployments, as well as online deployments.

1. Delete probe and gateway instances:

```
cloudctl case launch  \
  --case $CASEPATH    \
  --namespace $NAMESPACE     \
  --inventory netcoolIntegrationsOperator   \
  --action delete-custom-resources \
  --args "--crFile $CRFILE" \
  --tolerance 1
```

   Where `$CRFILE` variable is the path to the custom resource file that was use to create the instance and $CASEPATH is the path to the CASE archive in your local file system.

2. Delete the operator:

```
cloudctl case launch  \
  --case $CASEPATH    \
  --namespace $NAMESPACE     \
  --inventory netcoolIntegrationsOperatorSetup   \
  --action uninstall-operator  \
  --args "--inputDir $PWD/case" \
  --tolerance 1
```

   Optionally, you can also set the `--deleteOperatorGroup true` option to also remove the OperatorGroup resource when deleting the operator. For example:

```
cloudctl case launch  \
  --case $CASEPATH    \
  --namespace $NAMESPACE     \
  --inventory netcoolIntegrationsOperatorSetup   \
  --action uninstall-operator  \
  --args "--inputDir $PWD/case --deleteOperatorGroup true" \
  --tolerance 1
```

3. To uninstall the catalog source, run the following command:

```
 cloudctl case launch  \
  --case $CASEPATH  \
  --namespace $NAMESPACE     \
  --inventory netcoolIntegrationsOperatorSetup   \
  --action uninstall-catalog  \
  --args "--inputDir $PWD/case" \
  --tolerance 1
```

Using the CLI (native)

This section describes how to delete the operator using the CLI (native).

You can remove the operator deployment using the custom resource file using the following command: `kubectl delete -f deploy/operator.yaml`

Alternatively, you can remove the operator deployment using the CLI by specifying the deployment name: `kubectl delete deployment netcool-integrations-operator --namespace <NAMESPACE>`

Limitations

This section describes various limitations that currently exist.

The IBM Netcool Operations Insight Event Integrations Operator has the following limitations:

- Only the AMD64 / x86_64 architecture is supported.
- Prometheus integration is verified on Red Hat OpenShift Container Platform 4.5 and 4.6.
- There are several known limitations when enabling secure connection between probe clients and server:
  - The required files in the secret must be created using the `nc_gskcmd` utility.
  - If your ObjectServer is configured with FIPS 140-2, the password for the key database file (`omni.kdb`) must meet the requirements stated in IBM Documentation: https://www.ibm.com/support/knowledgecenter/SSSHTQ_8.1.0/com.ibm.netcool_OMNIbus.doc_8.1.0/omnibus/wip/install/task/omn_con_ssl_creatingkeydatabasefips.html.
  - When encrypting a string value using the encryption config key file (`encryption.keyfile`), you must use the AES_FIPS as the cipher algorithm. AES algorithm is not supported.
  - When connecting to an ObjectServer in the same cluster, you can connect the gateway to the secure connection proxy which is deployed with the IBM Netcool Operations Insight ObjectServer to encrypt the communication using TLS but the TLS termination is done at the proxy. It is recommended to enable Mutual Transport Layer Security (mTLS) in Service Mesh to encrypt cluster data network traffic. For more information, refer to the ServiceMeshControlPlane parameters section in the Service Mesh installation, usage, and release notes.
- There is no upgrade path from probe or gateway Helm charts to this Operator. Upgrades and rollback from Helm charts to Operator is not supported. Users must migrate the Helm release configuration to the respective probe or gateway custom resource YAML, deploy the custom resource YAML to install the probe or gateway, verify the integration is successful and working as expected before deleting the old Helm release.
- Ensure that your cluster has internet access for the operator to pull the container images from entitled registry and public registry.
- For the JDBC Gateway, if multiple deployments are connected to the same ObjectServer, in the event of a rolling update the deployments may result in the gateway attempting to execute insert, update, delete or duplicate events into the target RDBMS database. To address this limitation, the number of replicas for the Pod is limited to one and the recreate strategy is defined for the deployment.
- Currently the JDBC Gateway can only be provisioned for Audit mode.
- When the Generic Webhook Probe is configured to send HTTP requests to the target event source and there are more than one pod replica running, there may be duplicated HTTP requests sent to the target event source and duplicate events received in the ObjectServer. To avoid event duplication, limit the probe pod to one by setting the `probe.replicaCount` to 1 and `probe.autoscaling.enabled` to `false`.
- If you opt to use the OLM UI to install or configure the Gateway for Kafka, you must do so using YAML view instead of the Form view. There is a known limitation in the Form view whereby it changes the Kafka Client Producer Configuration from a multi-line string into a single-line string which is not expected by the gateway and the text field does not accept new line characters.
- Currently the Gateway for Kafka only supports `SASL/PLAIN` which is a simple username/password authentication mechanism. The gateway does not support `SASL/GSSAPI` which uses Kerberos for authentication.

Troubleshooting

The following table describes how to troubleshoot issues when deploying the operator and how to resolve them.

Table 7. Problems

Problem	Cause	Resolution
Probe logs show an error when loading or reading rules files. Failed during field verification check. Fields `SiteName` and `ScopeID` not found	The OMNIbus ObjectServer event grouping automation is not installed, hence the required fields are missing.	Install the event grouping automation in your ObjectServer and redeploy the probe or gateway.
The following error occurred when deploying the probe with PodDisruptionBudget enabled: `poddisruptionbudgets.policy is forbidden`	The user deploying the probe does not have the correct role to deploy the probe with **`PodDisruptionBudget`** enabled.	Administrator or Cluster Administrator role is required to deploy the probe with **`PodDisruptionBudget`** enabled.
The probe deployment failed to mount the ConfigMaps or some object name appeared to be somewhat random.	If a long instance name is used, the operator will generate a random suffix for objects that exceeds the character limit. This might cause mapping issues between the Kubernetes objects.	Use a shorter instance name, namely less than 20 characters.
Pod failed to mount the ConfigMaps or some object name appear to be somewhat random.	If a long name is used, the operator will generate a random suffix for objects that exceeds the character limit. This might cause mapping issues between the Kubernetes objects.	Use a shorter name, namely less than 20 characters.

Table 7. Problems (continued)

Problem	**Cause**	**Resolution**

In an offline or airgap installation mode, an error may occur when trying to mirror container images in an internal registry if there is already an existing repository for any of the images. If the following error is shown when pushing the container images to your existing internal registry, it is likely due to the recent changes made in the CASE inventory resources.yaml file.
 	
`api.sqaocp4611.cp.fyre.ibm.com:5000 cp/noi-int/ibm-netcool-integrations-operator         blobs=0 mounts=0 manifests=1 shared=0`
`info: Planning completed in 1.51s`
`sha256:9f53d5e340fc0486a2f0cd3130bd72733f0420bbcc054b19400c7fcbd3a6763a`
 	
`api.sqaocp4611.cp.fyre.ibm.com:5000/cp/noi-int/ibm-netcool-integrations-operator-bundle:1.2.0-amd64`
`error: unable to push manifest to api.sqaocp4611.cp.fyre.ibm.com:5000/cp/noi-int/netcool-integration-util:3.3.0-amd64: errors:`
`manifest blob unknown: blob unknown to registry`
`manifest blob unknown: blob unknown to registry`
`manifest blob unknown: blob unknown to registry`
`manifest blob unknown: blob unknown to registry`
`manifest blob unknown: blob unknown to registry`
	There is an update to the CASE inventory `resources.yaml` files where the media type of all container images has been updated to `application/vnd.docker.distribution.manifest.v2` from `application/vnd.oci.image.index.v1` for consistency with other IBM container images. The media type affects which digest the image registry returns which may cause an error if you try to push the new images to an existing registry with the old media type.	Delete the existing images in the internal registry or recreate the registry by running the CASE launch action (mirror-images) to initialize a new registry and then push the images into it.

Pod state changes to `CrashLoopBackOff` during AIOps installation and the logs shows an error connecting to the ObjectServer.	The IBM Cloud Pak for AIOps component may take some time to be ready. During this period, the ObjectServer pod may still be initializing and any probe or gateway that attempts to make connection will fail.	After all ObjectServer pods are ready and the services are accessible, the failing probe or gateway pods should automatically be restarted by Kubernetes. You can delete the probe or gateway pods to force Kubernetes to recreate the pod.

Table 7. Problems (continued)

Problem	**Cause**	**Resolution**

No pods created	In the event that there is no pod running after changing the SCC or the `setUIDandGID` in the custom resources, then it is likely that the pod was rejected because it did not match any SCC.	Check the `ReplicaSet` resource events:
		`oc describe replicaset <replicaset-name>`
		If you see an error like the one below, it means that the pod did not match the `restricted` SCC because it specifies a unique UID `1001` and Group ID `2001`.
		`Error creating: pods "yjprometheusprobe-ibm-netcool-probe-prometheusprobe-86b88bb9c5-" is forbidden: unable to validate against any security context constraint: [provider restricted: .spec.securityContext.fsGroup: Invalid value: []int64{2001}: 2001 is not an allowed group spec.containers[0].securityContext.runAsUser: Invalid value: 1001: must be in the ranges: [1000630000, 1000639999]]`
		Check that the custom resource has been updated and `setUIDandGID` parameter is set to `false`.

Pulsar Gateway pods goes into `CrashLoopBackOff` state due to `RuntimeException: Failed to load config into existing configuration data` error in initialization stage.	In `PulsarGateway` version 1.1.0, the Apache Pulsar Java client is upgrade to version 2.10 which introduces a condition to the Pulsar producer which requires `maxPendingMessagesAcrossPartitions >= maxPendingMessages`. When upgrading `PulsarGateway` operand version from `1.0.0` to `1.1.0` the default value of `maxPendingMessages` is `1000` while the new defaults for `maxPendingMessages` and `maxPendingMessagesAcrossPartitions` causing the error because the condition is not met.	When upgrading to version 1.1.0, update the `maxPendingMessages` value to `0` or add `maxPendingMessagesAcrossPartitions` to be greater or equal to the `maxPendingMessages` value in the `producer.configs` of the `PulsarGateway` custom resource.

Table 7. Problems (continued)

Problem	**Cause**	**Resolution**
Probe logs shows "Error: E-JPR-000-000: Exception caught in MultiChannelHttpRequestHandler. Channel Id: 494e4799 Channel Status: active=false Cause: javax.net.ssl.SSLHandshakeException: java.security.cert.CertificateException: No name matching <hostname> found Message: javax.net.ssl.SSLHandshakeException: java.security.cert.CertificateException: No name matching <hostname> found LocalMessage: javax.net.ssl.SSLHandshakeException: java.security.cert.CertificateException: No name matching <hostname> found"	Probe failed to match the Common Name (CN) of the certificate with the hostname of the target host when hostname verification is enabled.	Verify the Common Name (CN) in the certificate you intend to use to connect to the target host. If the CN does not match the hostname of the target host, recreate a new certificate and the CN in the certificate must match the hostname of the target host. Then, refer to "Sending Secure HTTP Requests to Event Resource" on page 129 to recreate the secret.
Probe logs shows "Error: E-JPR-000-000: Failed to connect; ProbeException: Fail to start Transport module for connection; TransportAuthorizeException: java.security.cert.CertificateException: No subject alternative names present" or "java.security.cert.CertificateException: No subject alternative names matching <hostname> found".	Probe failed to match the Subject Alternate Name (SAN) of the server certificate with the hostname of the target host when hostname verification is enabled.	Recreate the server certificate and the SAN must match the hostname of the target host. Then, refer to "Sending Secure HTTP Requests to Event Resource" on page 129 to recreate the secret.

Known issues

This section describes the known issues that currently exist.

Operator installation hangs when using the OCP Web console

If you observe that the operator installation via the OCP Web console hangs either in Upgrading or Installing state indefinitely, this could be due to a problem in the Operator Catalog Source in resolving one or more images required by the Operator Lifecycle Manager (OLM).

Workaround:

Download the PPA archive from Passport Advantage® and follow the steps to install the operator using the CLI.

PodDisruptionBudget resource is not created by the operator after installing a probe custom resource

In newer OCP versions, such as OCP 4.8, the PodDisruptionBudget resource is not created when the probe custom resource is configured with `poddisruptionbudget.enabled: true`. This may occur because the operator tries to create a PodDisruptionBudget resource using and older `policy/v1beta1` API version which is deprecated in newer OCP version such as OCP 4.8.

Workaround:

You can create a PodDisruptionBudget resource manually by selecting the probe pod `app.kubernetes.io/instance: <instance name>` label such as below:

```
apiVersion: policy/v1
kind: PodDisruptionBudget
metadata:
  name: myprobe-pdb
spec:
  minAvailable: 1
  selector:
    matchLabels:
      app.kubernetes.io/instance: myprobe
```

Pulsar probe continues to run after a disconnection has occurred

In some cases, the Pulsar probe continues to run after the probe has disconnected from the Pulsar broker. The probe debug log shows that it has closed its Pulsar consumer resources, but subsequent heartbeat checks do not detect that the Pulsar client thread has already closed.

```
Debug: D-JPR-000-000: [PulsarConsumerRunnable] - Pulsar consumer thread ends.
Debug: D-JPR-000-000: [PulsarConsumerRunnable] - Cleaning up Pulsar consumer resources
Debug: D-JPR-000-000: [PulsarConsumerRunnable] - Closing Pulsar consumer
Debug: D-JPR-000-000: [PulsarConsumerRunnable] - Cleaning up Pulsar client resources
Debug: D-JPR-000-000: [PulsarConsumerRunnable] - Closing Pulsar client
...
Debug: D-JPR-000-000: com.ibm.tivoli.netcool.omnibus.probe.framework.HeartbeatTask.run ENTERING
Information: I-JPR-000-000: Checking Probe Heartbeat
Debug: D-JPR-000-000: com.ibm.tivoli.netcool.omnibus.probe.framework.HeartbeatTask.run EXITING
Debug: D-JPR-000-000: com.ibm.tivoli.netcool.omnibus.probe.framework.HeartbeatTask.run ENTERING
Information: I-JPR-000-000: Checking Probe Heartbeat
Debug: D-JPR-000-000: com.ibm.tivoli.netcool.omnibus.probe.framework.HeartbeatTask.run EXITING
```

Workaround:

You can set the Pulsar probe `probe.inactivity` property to the period (in seconds) for the probe to detect that there is no activity or Pulsar messages received from the topic before disconnecting the probe to allow Kubernetes to restart the pod and reestablish the Pulsar connection with the broker.

JDBC Gateway shows an error status in the Persistent Volume Claim resource phase

When the JDBC Gateway is configured with dynamic volume provisioning, the operator creates a Persistent Volume Claim (PVC) resource during the first initialization stage to claim a persistent volume. If the JDBC Gateway custom resource specification is updated, the operator will reconcile the custom resource and attempt to update the PVC resource which will cause the `status.phase` of the PVC to change to `Error` because the some of the PVC resource specifications are immutable.

```
    - kind: PersistentVolumeClaim
      status:
        phase: Error
```

The operator pod log will log the following error:

```
Forbidden: spec is immutable after creation except resources.requests for bound claims
```

The JDBC Gateway will continue to run and you can safely ignore this error.

Workaround:

Use a static provisioning by pre-creating a PVC resource before installing the JDBC Gateway. In the JDBC Gateway custom resource, set the `persistence.existingClaimName` property to the name of the pre-created PVC.

JDBC Gateway throws an error when inserting a duplicate event into journal table

The exception shown at the end of this section is printed in the pod logs in the following scenario:

1. You install the gateway and wait for the gateway to replicate events.
2. Events are replicated in the gateway.
3. You restart the gateway (old events are still in the ObjectServer).
4. The gateway throws the exception shown below.

```
21-09-03T03:34:37: Error: E-GJA-000-000: [ngjava]: G_JDBC: pool-1-thread-2: In logException
2021-09-03T03:34:37: Error: E-GJA-000-000: [ngjava]: G_JDBC: pool-1-thread-2: Duplicate entry '1090171-AGG_P-2021-09-01 07:40:09-2' for key 'PRIMARY'
2021-09-03T03:34:37: Error: E-GJA-000-000: [ngjava]: G_JDBC: pool-1-thread-2: java.sql.BatchUpdateException: Duplicate entry '1090171-AGG_P-2021-09-01
07:40:09-2' for key 'PRIMARY'
2021-09-03T03:34:37: Error: E-GJA-000-000: [ngjava]: G_JDBC: pool-1-thread-2:   sun.reflect.NativeConstructorAccessorImpl.newInstance0(Native Method)
2021-09-03T03:34:37: Error: E-GJA-000-000: [ngjava]: G_JDBC: pool-1-thread-2:
sun.reflect.NativeConstructorAccessorImpl.newInstance(NativeConstructorAccessorImpl.java:83)
2021-09-03T03:34:37: Error: E-GJA-000-000: [ngjava]: G_JDBC: pool-1-thread-2:
sun.reflect.DelegatingConstructorAccessorImpl.newInstance(DelegatingConstructorAccessorImpl.java:57)
2021-09-03T03:34:37: Error: E-GJA-000-000: [ngjava]: G_JDBC: pool-1-thread-2:   java.lang.reflect.Constructor.newInstance(Constructor.java:437)
2021-09-03T03:34:37: Error: E-GJA-000-000: [ngjava]: G_JDBC: pool-1-thread-2:   com.mysql.cj.util.Util.handleNewInstance(Util.java:192)
2021-09-03T03:34:37: Error: E-GJA-000-000: [ngjava]: G_JDBC: pool-1-thread-2:   com.mysql.cj.util.Util.getInstance(Util.java:167)
2021-09-03T03:34:37: Error: E-GJA-000-000: [ngjava]: G_JDBC: pool-1-thread-2:   com.mysql.cj.util.Util.getInstance(Util.java:174)
2021-09-03T03:34:37: Error: E-GJA-000-000: [ngjava]: G_JDBC: pool-1-thread-2:
com.mysql.cj.jdbc.exceptions.SQLError.createBatchUpdateException(SQLError.java:224)
2021-09-03T03:34:37: Error: E-GJA-000-000: [ngjava]: G_JDBC: pool-1-thread-2:
com.mysql.cj.jdbc.ClientPreparedStatement.executeBatchSerially(ClientPreparedStatement.java:853)
2021-09-03T03:34:37: Error: E-GJA-000-000: [ngjava]: G_JDBC: pool-1-thread-2:
com.mysql.cj.jdbc.ClientPreparedStatement.executeBatchInternal(ClientPreparedStatement.java:435)
2021-09-03T03:34:37: Error: E-GJA-000-000: [ngjava]: G_JDBC: pool-1-thread-2:   com.mysql.cj.jdbc.StatementImpl.executeBatch(StatementImpl.java:796)
2021-09-03T03:34:37: Error: E-GJA-000-000: [ngjava]: G_JDBC: pool-1-thread-2:
com.ibm.tivoli.netcool.integrations.jdbc.GWJdbcBatchProcessor.processRows(GWJdbcBatchProcessor.java:518)
2021-09-03T03:34:37: Error: E-GJA-000-000: [ngjava]: G_JDBC: pool-1-thread-2:
com.ibm.tivoli.netcool.integrations.jdbc.GWJdbcBatchProcessor.run(GWJdbcBatchProcessor.java:350)
2021-09-03T03:34:37: Error: E-GJA-000-000: [ngjava]: G_JDBC: pool-1-thread-2:
java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1160)
2021-09-03T03:34:37: Error: E-GJA-000-000: [ngjava]: G_JDBC: pool-1-thread-2:
java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:635)
2021-09-03T03:34:37: Error: E-GJA-000-000: [ngjava]: G_JDBC: pool-1-thread-2:   java.lang.Thread.run(Thread.java:822)
```

After the gateway has restarted, it will attempt to replicate all events from the ObjectServer into the database. Events that were already replicated to the database cannot be re-inserted again due to the primary key unique constraint of the database table. These exceptions will not cause the gateway process to prematurely end; the gateway will continue replicating all events from the object server. These exceptions will continue to be printed in the logs until the gateway processes new events that were not replicated earlier. These new events will not produce this error and will be inserted successfully into the database.

Operand images are not automatically upgraded by the operator after upgrading from v3.1 to v3.2

If you have installed the operator version 3.1.0 or 3.1.1 and upgraded to version 3.2.3, you may notice that the operator does not upgrade the container images automatically even when the `useDefaultOperandImages` parameter is set to `true` in your custom resource specification.

For version 3.1.0 and 3.1.1 of the operator the operand images start with the prefix `event` and in operator version 3.2.0 or higher the operand images start with `netcool`.

Workaround

To resolve the issue, update the image name in your custom resource specification to change the image prefix from `event` to `netcool`. For example, change `event-probe-messagebus` to `netcool-probe-messagebus`. Apply the changes to the custom resource and observe the container image is updated in the pod. Note that if the `useDefaultOperandImages` parameter is set to `false` in your custom resource specification, then you must also update the image digest or tag to the latest version too.

You may query the image name and digest from the operator `olm.relatedImage` annotation using the following command:

```
 oc get pod <netcool-integations-operator-podname>  -o go-template --template='{{ range
$key,$value := .metadata.annotations }}{{ $key }}{{ ": " }}{{ $value }}{{ "\n" }}{{ end }}' |
grep 'relatedImage'
```

Where <netcool-integations-operator-podname> is the operator pod name.

For example:

```
$ oc get pod netcool-integrations-operator-588b4cbb6f-wzsp9    -o go-template
--template='{{ range $key,$value := .metadata.annotations }}{{ $key }}{{ ": " }}{{ $value }}
{{ "\n" }}{{ end }}' | grep 'relatedImage'
olm.relatedImage.netcool-gateway-cem: cp.icr.io/cp/noi-int/netcool-gateway-
cem@sha256:78f06af652e405ce199737c323a0643a4d64acc015f6ac2eff40691f066a6a64
olm.relatedImage.netcool-gateway-jdbc: cp.icr.io/cp/noi-int/netcool-gateway-
jdbc@sha256:339c8bd3673bde3e6bc4a4d0c3248f26217932296e204a6aadd7a31bcdb0790d
olm.relatedImage.netcool-gateway-messagebus: cp.icr.io/cp/noi-int/netcool-gateway-
messagebus@sha256:d4c2206dd3f1d4ae2b8f65d25735dd324ad9b4c91518bfd3bb0927533830372d
olm.relatedImage.netcool-integration-util: cp.icr.io/cp/noi-int/netcool-integration-
util@sha256:5457ef0df25856d84f3a08a1310c7174014f4cf4b023f6ce784126dc4538e153
olm.relatedImage.netcool-probe-messagebus: cp.icr.io/cp/noi-int/netcool-probe-
messagebus@sha256:8c9a95211cbb8811d6c7f1f42063dac64d4c6010a0c5dc7313332a706dd18a4a
```

Operand images are not automatically upgraded by the operator after upgrading from v3.2 to v3.3 when using Operator Lifecycle Manager (OLM)

When upgrading to operator version 3.3.0 , your operand (probe or gateway) images may not be upgraded to the latest version even when the `useDefaultOperandImages` parameter is set to `true` in your custom resource specification.

This is because in operator v3.3.0, the operator's leader election has changed from leader-for-life to leader-with-lease mode. During the operator deployment rolling update, there may be two operator controllers that is acting as a leader (also called split-brain) for a short period of time as the Config Map resource used in the leader election and the mechanism to determine if an existing leader is active has changed.

Workaround

To avoid the contention between two operator pods during rolling update, you can opt uninstall your existing operator before installing the version 3.3.0 operator. This will stop the old operator pod and the new operator pod will become the leader after the installation of version 3.3.0.

However, if you have already performed an upgrade, you can still let the new operator version upgrade your operand images

by restarting the operator controller either by deleting the operator pod, or scaling down the `netcool-integrations-operator` deployment to 0 pods and then scale it back to 1 pod replica. When the operator pod is running, it will reconcile all existing custom resource which will trigger an image upgrade if necessary.

ibm-cloud-pak plugin V1.11.0 failing its prerequisite check

There is a known issue whereby the `ibm-cloud-pak` plugin V1.11.0 fails its prerequisites check even when all Kubernetes prerequisites are met. You will see the following message in the error log after running the install command:

```
Welcome to the CASE launcher
Attempting to retrieve and extract the CASE from the specified location
[√] CASE has been retrieved and extracted
Attempting to validate the CASE
[√] CASE has been successfully validated
Attempting to locate the launch inventory item, script, and action in the specified CASE
[√] Found the specified launch inventory item, action, and script for the CASE
Attempting to check the cluster and machine for required prerequisites for launching the item
Checking for required prereqs...

Prerequisite                                                          Result
OpenShift Container Platform Kubernetes version is 1.23.0 or greater  true
Cluster has at least one amd64 node                                   false
Client has oc installed                                               true
Client has cloudctl installed                                         true

Required prereqs result: FAILED
```

```
See the logs for more details. Searching for 'Failed to validate prereq item' in the logs will
show errors encountered while evaluating the prereq conditions

Additional information on prereqs for the installCatalog action
================================================================
The setup action must be run on one of OpenShift Container
Platform on amd64 Linux. The minimum level of Kubernetes on
each platform are described in the CASE prerequisites.
The client must have oc installed to execute the launcher script.

Error: Parsing the actions prereqs failed
Error: * An error was encountered executing process
    - Command: bash
    - Args: -c /root/.ibm-pak/data/launch/ibm-netcool-integrations/3.8.0/
netcoolIntegrationsOperatorSetup/main-launch.sh
    - Err: exit status 1
    - Output:
```

Workaround

There is no workaround for `ibm-cloud-pak` plugin V1.11.0, You must install `ibm-cloud-pak` plugin V1.11.1, running with the `--insecure` flag:

```
  oc ibm-pak launch \
   $CASE_NAME \
     --version $CASE_VERSION \
     --action install-catalog \
     --inventory $CASE_INVENTORY_SETUP \
     --namespace $NAMESPACE \
     --args "--catalogImage icr.io/cpopen/ibm-netcool-integrations-operator-catalog:3.8.0-amd64
--recursive \
     --inputDir ~/.ibm-pak/data/cases/$CASE_NAME/$CASE_VERSION" --insecure
```

Multibyte Encoding

Multibyte characters have issues with encoding in Operator 3.11.0.

The Probe for Kafka integration is not receiving multibyte characters correctly.

Multibyte characters have issues being displayed in Event Lists.

Chapter 2. Deploying integrations

For details of the integrations that you can deploy using the IBM Netcool Operations Insight Event Integrations Operator, see the following sections:

- "Deploying the Gateway for JDBC Integration" on page 65
- "Deploying the Probe for Prometheus Integration" on page 75
- "Deploying the Probe for Kafka Integration" on page 80
- "Deploying the Gateway for Kafka" on page 100
- "Deploying the Probe for Pulsar Integration" on page 107
- "Deploying the Gateway for Pulsar Integration" on page 115
- "Deploying the Generic Webhook Probe" on page 119
- "Deploying the Probe for IBM SevOne NPM Integration" on page 140
- "Deploying the Probe for Turbonomic Integration" on page 144

Connecting with IBM Cloud Pak for AIOps

This section outlines the steps required to configure probes and gateways to integrate with IBM Cloud Pak for AIOps.

Connecting on-premise probes and gateways to IBM Cloud Pak for AIOps

This section describes how to configure on-premise integrations.

Connecting on-premise probes

1. The probe will make a connection to the TLS proxy of IBM Cloud Pak for AIOps.
2. Get details of the proxy port connection and its secret on AIOPs.

```
oc get service | grep proxy
   evtmanager-proxy              NodePort       172.30.44.167    <none>     6001:30455/
TCP,6002:31823/TCP

oc get secret | grep proxy
evtmanager-proxy-tls-secret     kubernetes.io/tls                     2      15
```

   In the example, the NodePort for the primary ObjectServer is 30455 and the NodePort for the backup ObjectServer is 31823.
3. Import the proxy `tls` secret to `omni.kdb` at the on-premise server.

```
kubectl get secrets evtmanager-proxy-tls-secret -o yaml -n cp4waiops | egrep "tls.crt:" |
awk '{print $2}' | base64 --decode  > emproxy.cem
$NCHOME/bin/nc_gskcmd -keydb -create -db "$NCHOME/etc/security/keys/omni.kdb" -pw password
-stash -expire 366
$NCHOME/bin/nc_gskcmd -cert -add -file emproxy.cem -db $NCHOME/etc/security/keys/omni.kdb
-stashed
$NCHOME/bin/nc_gskcmd -cert -list -db $NCHOME/etc/security/keys/omni.kdb -pw password
```

4. Edit `$NCHOME/etc/omni.dat` and add the proxy services.

   For example `proxyport` from Step 2.

```
[AGG_P]
{
    Primary:       evtmanager-proxy.cp4waiops.svc ssl proxyport
}
```

5. Edit `/etc/hosts` and add the proxy service details.

   IP address : Cluster master node IP address.

   ```
   9.30.138.122 evtmanager-proxy.cp4waiops.svc
   ```

6. Configure the probe to connect to the servers set up in the previous step.

Connecting on-premise gateways

1. The gateway will make a connection to the TLS proxy of IBM Cloud Pak for AIOps.
2. Get details of the proxy port connection and its secret on AIOPs.

   ```
   oc get service | grep proxy
      evtmanager-proxy             NodePort       172.30.44.167    <none>   6001:30455/
   TCP,6002:31823/TCP

   oc get secret | grep proxy
   evtmanager-proxy-tls-secret
   kubernetes.io/tls                    2      15
   ```

3. Import the proxy `tls` secret to `omni.kdb` at on premise server.

   ```
   kubectl get secrets evtmanager-proxy-tls-secret -o yaml -n cp4waiops | egrep "tls.crt:" |
   awk '{print $2}' | base64 --decode  > emproxy.cem
   $NCHOME/bin/nc_gskcmd -keydb -create -db "$NCHOME/etc/security/keys/omni.kdb" -pw password
   -stash -expire 366
   $NCHOME/bin/nc_gskcmd -cert -add -file emproxy.cem -db $NCHOME/etc/security/keys/omni.kdb
   -stashed
   $NCHOME/bin/nc_gskcmd -cert -list -db $NCHOME/etc/security/keys/omni.kdb -pw password
   ```

4. Edit `$NCHOME/etc/omni.dat` and add the proxy services.

   For example `proxyport` from Step 2.

   ```
   [AGG_P]
   {
       Primary:       evtmanager-proxy.cp4waiops.svc ssl proxyport
   }
   ```

5. Edit `/etc/hosts` and add the proxy service details. Three entries are required for the gateway connection to work, IP address of the AIOPS Cluster, svc name and nodeport service name.

   ```
   oc get services | grep evtmanager

   Nodeport:
   evtmanager-objserv-agg-primary-nodeport                          NodePort
   172.30.164.99    <none>
   4100:31933/TCP,30102:30102/TCP                           30h
   ```

   `/etc/hosts`:

   ```
   9.30.138.122 evtmanager-proxy.cp4waiops.svc evtmanager-objserv-agg-primary-nodeport
   ```

   Note: The `evtmanager-objserv-agg-primary-nodeport` hostname is required for IDUC connection by the gateway.

6. Configure the gateway to connect to the servers set up in the previous step.

Connecting probes and gateways on Red Hat OpenShift

This section describes how to configure probes and gateways on Red Hat OpenShift.

Connecting the probe to IBM Cloud Pak for AIOps in the same namespace

1. Select the project where AIOps is installed. Go to **Installed Operators**.
2. Choose **IBM Netcool Operations Insight Event Integrations**.

3. Create an **Instance** in the namespace, for example: **Probe for Prometheus Integration**.
4. Go to **Yaml** view. Example settings are shown below. view and configure the object server parameters under the `netcool` parameter according to the connection types described below.

Default Connection

1. To identify the ObjectServer service details, run the following command:

```
kubectl get services | grep evtmanager | grep -e 'objserv-agg' | grep -v 'nodeport' | awk
'{printf "%s\t%s\n", $1, $5}'
evtmanager-objserv-agg-backup    4100/TCP,4300/TCP
evtmanager-objserv-agg-primary    4100/TCP
```

2. Configure your probe ObjectServer configuration in the CR as shown in the example below:

```
netcool:
  backupHost: evtmanager-objserv-agg-backup
  backupPort: 4300
  backupServer: AGG_B
  connectionMode: default
  primaryHost: evtmanager-objserv-agg-primary
  primaryPort: 4100
  primaryServer: AGG_P
```

SSLAndAuth Connection

1. `SSLAndAuth` connection requires a secret with a key database file containing the TLS certificate and credentials to authenticate with the ObjectServer. This secret must be pre-created and the name of the secret is set as the `spec.netcool.secretName` value.
2. Run the `create-noi-secret.sh` script to create the secret. Get this script from the CASE bundle as described below:

```
cloudctl case save --case https://github.com/IBM/cloud-pak/blob/master/repo/case/ibm-netcool-
integrations/1.2.0/ibm-netcool-integrations-1.2.0.tgz?raw=true -t 1 --outputdir /root/casedir

gunzip ibm-netcool-integrations-1.2.0.tgz
tar -xvf ibm-netcool-integrations-1.2.0.tar
cd ibm-netcool-integrations/inventory/ibmNetcoolGatewayCEMSetup/files
```

3. Fill up the appropriate values in the `create-noi-secret.config` file:

```
IMAGE_NAME : # Either specify full path to netcool probe image name or pull the image
manually in your server before
             running the script
NAMESPACE = # specify your namespace
KEY_DATABASE_PASSWORD = #Specify any password
TLS_ENABLED = set to true
SECRET_NAME = specify the secretname which will be created in your namespace which will be
filled up at spec.netcool.secretName
            e.g SECRET_NAME = aiopssecret (for illustration purposes)
AUTH_USERNAME=root
AUTH_PASSWORD= # check the password using the command below
kubectl get secret evtmanager-omni-secret  -o json -n yournamespace | grep
OMNIBUS_ROOT_PASSWORD  | cut -d : -f2 | cut -d '"' -f2 | base64 -d;
NOI_RELEASE_NAME=evtmanager
NOI_NAMESPACE= # your namespace
```

4. Run the script `./create-noi-secret.sh`
5. Check whether your secret from SECRET_NAME from Step 3 was created in your chosen namespace.
6. To identify the ObjectServer service details, run the following command:

```
kubectl get services | grep evtmanager | grep 'proxy' | awk '{printf "%s\t%s\n", $1, $5}'

evtmanager-proxy                        6001:30455/TCP,6002:31823/TCP
```

7. Configure your probe ObjectServer configuration in the CR as shown in the example below:

```
  netcool:
     connectionMode: SSLAndAuth
     primaryPort: 30455             # Proxy primary Nodeport number
     backupPort: 31823              # Proxy secondary Nodeport port
     primaryServer: AGG_P
     backupServer: AGG_B
     primaryHost: 10.17.73.66         # Master node IP
     backupHost: 10.17.73.66         # Master node IP
     secretName: aiopssecret
   probe:
     sslServerCommonName: evtmanager-proxy.yournamespace.svc
```

Connecting the gateway to IBM Cloud Pak for AIOps in the same namespace

SSLAndAuth Connection

1. `SSLAndAuth` connection requires a secret with a key database file containing the TLS certificate and credentials to authenticate with the ObjectServer. This secret must be pre-created and the name of the secret is set as the `spec.netcool.secretName` value.
2. Run the `create-noi-secret.sh` script to create the secret. Get this script from the CASE bundle as described below:

```
cloudctl case save --case https://github.com/IBM/cloud-pak/blob/master/repo/case/ibm-netcool-
integrations/1.2.0/ibm-netcool-integrations-1.2.0.tgz?raw=true -t 1 --outputdir /root/casedir

gunzip ibm-netcool-integrations-1.2.0.tgz
tar -xvf ibm-netcool-integrations-1.2.0.tar
cd ibm-netcool-integrations/inventory/ibmNetcoolGatewayCEMSetup/files
```

3. Fill up the appropriate values in `create-noi-secret.config` file:

```
IMAGE_NAME : # Either specify full path to netcool gateway image name or pull the image
manually in your server before
             running the script

NAMESPACE = # specify your namespace
KEY_DATABASE_PASSWORD = #Specify any password
TLS_ENABLED = set to true
SECRET_NAME = specify the secretname which will be created in your namespace which will be
filled up at spec.netcool.secretName
            e.g SECRET_NAME = aiopssecret (for illustration purposes)

AUTH_USERNAME=root

AUTH_PASSWORD= # check the password using the command below
kubectl get secret evtmanager-omni-secret  -o json -n yournamespace | grep
OMNIBUS_ROOT_PASSWORD  | cut -d : -f2 | cut -d '"' -f2 | base64 -d;

NOI_RELEASE_NAME=evtmanager
NOI_NAMESPACE= # your namespace
```

4. Run the script `./create-noi-secret.sh`
5. Check whether your secret from SECRET_NAME from Step 3 was created in your desired namespace.
6. To identify the ObjectServer service details, run the following command:

```
kubectl get services | grep evtmanager | grep -e 'node\|proxy' | awk '{printf "%s\t%s\n",
$1, $5}'

evtmanager-objserv-agg-backup-nodeport    4100:30255/TCP,30165:30165/TCP
evtmanager-objserv-agg-primary-nodeport    4100:32343/TCP,31297:31297/TCP
evtmanager-proxy                        6001:30455/TCP,6002:31823/TCP
```

7. Configure the gateway ObjectServer connection details as shown in the example below:

```
  netcool:
     connectionMode: SSLAndAuth
     primaryPort: 30455             # Proxy primary Nodeport number
     backupPort: 31823              # Proxy secondary Nodeport port
     primaryServer: AGG_P
     backupServer: AGG_B
```

```
      primaryIP: 10.17.73.66         # Master node IP
      backupIP: 10.17.73.66          # Master node IP
      primaryHost: evtmanager-proxy.yournamespace.svc
      backupHost: evtmanager-proxy.yournamespace.svc
      primaryIDUCHost: evtmanager-objserv-agg-primary-nodeport
      backupIDUCHost: evtmanager-objserv-agg-backup-nodeport
      secretName: aiopssecret1
```

Deploying the Gateway for JDBC Integration

IBM Netcool/OMNIbus Gateway for JDBC processes events and alerts from IBM Netcool/OMNIbus ObjectServer and forwards them to a target Relational Database Management System (RDBMS) database.

Note: The IBM Netcool Operations Insight Event Integrations Operator must be deployed before deploying this integration. See "Installing the IBM Netcool Operations Insight Event Integrations Operator" on page 21. See also "Pre-installation tasks" on page 70

To deploy a new instance of the gateway, review the custom resource YAML template supplied with the operator package, update the parameters if required, and save the configurable parameters in a file, for example `gateways_v1_jdbcgateway.yaml`.

Then deploy the custom resource using the updated file:

```
oc apply -f gateways_v1_jdbcgateway.yaml --namespace <NAMESPACE>
```

Note: If you are installing using the CLI, you can get the custom resource Yaml file from the `ibm-netcool-integrations/inventory/netcoolIntegrationsOperatorSetup/files/op-cli/deploy/crs` directory in our CASE archive. If you are installing using the Operator Lifecycle Manager (OLM) Web console, the custom resource is provided in the YAML view.

The following table describes the configurable parameters for this gateway and lists their default values.

Configurable parameters

Parameter name	**Description**
spec.license.accept	The license state of the image being deployed. Enter `true` to install and use the image. The default value is `false`.
spec.version	Gateway application version. Accepted versions are `1.0.0` or `1.1.0`.
spec.helmValues.image.useDefaultOperandImages	Use default operand images. If `true`, the operator will use default images for the operands ignoring the image settings in the custom resource YAML file. The operator will perform an upgrade with the latest images from the operator. If `false`, the operator will use the images specified in the custom resource YAML file to deploy the operand. This should be set to `false` if you are pulling from a private repository and no image mirroring is configured in the cluster. Default is `true`.
spec.helmValues.image.gateway	Name of the gateway image. The image name must be `netcool-gateway-jdbc`. Default value is `cp.icr.io/cp/noi-int/netcool-gateway-jdbc@sha256:589da54c07baa0b9c3319706862f8ebe73768b27f96307811d6cb55bd80b97a0`.

Parameter name	Description
spec.helmValues.image.utility	The Utility image repository. The default is `cp.icr.io/cp/noi-int/netcool-integration-util@sha256:97beb12f8660cdc2392507e9e0d42c026b7ff41f3a5c811785621af8a3906e0a`.
spec.helmValues.image.pullPolicy	Image pull policy. The default is `IfNotPresent`.
spec.helmValues.global.image.secretName	Name of the Secret containing the Docker Config to pull image from a private repository. Leave blank if the gateway image already exists in the local image repository or the Service Account has been assigned with an Image Pull Secret. The default is `nil`.
spec.helmValues.global.serviceAccountName	Name of the service account to be used by the deployment. If the Cluster Administrator has already created a service account in the namespace, specify the name of the service account here. If left blank, the operator will automatically create a new service account in the namespace when it is deployed. This new service account will be removed from the namespace when the instance is deleted. The default is `nil`.
spec.helmValues.global.persistence.existingClaimName	Specify the name of the existing Persistent Volume Claim (PVC) to be used to store the JDBC Driver, if not set the operator will create a PVC using dynamic provisioning. The default is `nil`.
spec.helmValues.global.persistence.useDynamicProvisioning	Use Storage Class to dynamically create Persistent Volume and Persistent Volume Claim. The default is `true`.
spec.helmValues.global.persistence.storageClassName	Storage Class for dynamic provisioning. The default is `""`.
spec.helmValues.global.persistence.selector.label	The Persistent Volume Claim Selector label key to refine the binding process when dynamic provisioning is not used. The default is `""`.
spec.helmValues.global.persistence.selector.value	The Persistent Volume Claim Selector label value related to the Persistent Volume Claim Selector label key. The default is `""`.
spec.helmValues.global.persistence.storageSize	Storage size for the PVC. The default is `10Mi`.
spec.helmValues.global.persistence.accessMode	Specifies the Access Mode for the PVC. The default is `ReadWriteMany`.

Parameter name	Description
spec.helmValues.netcool.connectionMode	The connection mode to use when connecting to the Netcool/OMNIbus ObjectServer. Refer to Securing probe and ObjectServer communications for more details. **Note**: Refer to limitations section for more details on available connection modes for your environment. Requires a pre-created secret containing files/values for secured communication. The default is `AuthOnly`.
spec.helmValues.netcool.primaryServer	The primary Netcool/OMNIbus server the gateway should connect to (required). Usually set to NCOMS or AGG_P. The default is `AGG_P`.
spec.helmValues.netcool.primaryHost	The host of the primary Netcool/OMNIbus Objsect Server hostname (required). Specify the ObjectServer Hostname. The default is `nil`.
spec.helmValues.netcool.primaryIP	The primary Netcool/OMNIbus ObjectServer IP address. If specified along with primaryHost, a host alias entry will be added. The default is `nil`.
spec.helmValues.netcool.primaryPort	The port number of the primary Netcool/OMNIbus ObjectServer (required). The default is `4100`.
spec.helmValues.netcool.primaryIDUCHost	The primary Netcool/OMNIbus ObjectServer IDUC Host or Service name. Should be set if the primary IDUC host is different from the primary ObjectServer hostname.When connecting to NOI on OCP, this should be set to the primary ObjectServer NodePort service name. The default is `nil`.
spec.helmValues.netcool.backupServer	The backup Netcool/OMNIbus ObjectServer to connect to. If the backupServer, backupHost and backupPort parameters are defined in addition to the primaryServer, primaryHost, and primaryPort parameters, the gateway will be configured to connect to a virtual ObjectServer pair called `AGG_V`. The default is `nil`.
spec.helmValues.netcool.backupHost	The host of the backup Netcool/OMNIbus ObjectServer. Specify the backup ObjectServer Hostname. The default is `nil`.
spec.helmValues.netcool.backupIP	The backup Netcool/OMNIbus ObjectServer IP address. If specified along with primaryHost, a host alias entry will be added. The default is `nil`.
spec.helmValues.netcool.backupPort	The port of the backup Netcool/OMNIbus ObjectServer. The default is `nil`.
spec.helmValues.netcool.backupIDUCHost	The backup Netcool/OMNIbus ObjectServer IDUC Host or Service name. Should be set if the primary IDUC host is different from the primary ObjectServer hostname. When connecting to NOI on OCP, this should be set to the primary ObjectServer NodePort service name. The default is `nil`.

Parameter name	Description
spec.helmValues.netcool.secretName	A pre-created secret for AuthOnly, SSLOnly or SSLAndAuth connection mode. Certain fields are required depending on the connection mode. The default is `nil`.
spec.helmValues.netcool.waitTimeout	The duration (in minutes) to wait for the ObjectServer to become available. The default is `0` (disabled).
spec.helmValues.jdbcgateway.setUIDandGID	When set to `true`, the operator will specify the UID and GID values for the netcool user else the netcool user will not be assigned any UID or GID by the operator. When deploying in an environment that is enforced by SecurityContextConstraints (SCC) that requires the process to have a fixed UID and GID value, this parameter must be set to `true`. Otherwise, if deploying in an environment enforced by an SCC that requires the process to support arbitrary UIDs, for example in the Openshift `restricted` SCC, this parameter must be set to `false`. Default value is `false`.
spec.helmValues.jdbcgateway.locale	Environment locale setting. Used as the LC_ALL environment variable. The default is `en_US.utf8`.
spec.helmValues.jdbcgateway.waitTimeout	Timeout duration (in minutes) for the initContainer instance to wait for the JDBC driver(s) to be copied into the pod. If set to `0`, the initContainer instance will not be started at all. The default is 5.
spec.helmValues.jdbcgateway.messageLevel	The gateway log message level. The default value is `warn`
spec.helmValues.jdbcgateway.debug	Enable/disable Java Gateway Reader/Writer debug messages. The default value is `false`
spec.helmValues.jdbcgateway.java.tlsSecretName	Name of the Secret containing the TLS certificate file in PEM format (tls.crt) for the target database. The default is `nil`.
spec.helmValues.jdbcgateway.java.enableOracleCiphers	Set to true to enable the IBM Java to use the Oracle cipher suite names, otherwise when set to false IBM Java will use the old cipher suite names. The default is `true`
spec.helmValues.jdbcgateway.java.javaArgs	Used to specify any additional Java Virtual Machine (JVM) arguments to the gateway. The default value is unset (` `)
spec.helmValues.jdbcgateway.java.debug	Enable/disable Java Gateway debug messages. The default is `false`
spec.helmValues.jdbcgateway.mapper.debug	Enable/disable mapper debug messages. The default is `false`
spec.helmValues.jdbcgateway.ngtk.debug	Enable/disable NGTK library debug messages. The default is `false`
spec.helmValues.jdbcgateway.rdrwtr.bufferSize	Size of the SQL command buffer. The default is `10`

Parameter name	Description
spec.helmValues.jdbcgateway.rdrwtr.debug	Enable/disable reader/writer debug messages. The default is `false`
spec.helmValues.jdbcgateway.rdrwtr.logOSSql	Log all SQL sent to ObjectServer in debug mode. The default is `false`
spec.helmValues.jdbcgateway.jdbc.credentialsSecretName	Name of the Secret containing the user credentials to connect to the target database. This field is mandatory.
spec.helmValues.jdbcgateway.jdbc.driver	Specifies the type of JDBC Driver. This field is mandatory.
spec.helmValues.jdbcgateway.jdbc.url	JDBC connection URL. This field is mandatory.
spec.helmValues.jdbcgateway.jdbc.initializationString	Specifies the JDBC connection initialization string. The default is `nil`
spec.helmValues.jdbcgateway.jdbc.mode	Specifies the JDBC gateway mode. This field is mandatory. The default is `AUDIT`
spec.helmValues.jdbcgateway.jdbc.statusTableName	Specifies the target table for alerts.status. This field is mandatory. The default is `status`
spec.helmValues.jdbcgateway.jdbc.journalTableName	Specifies the target table for alerts.journal. This field is mandatory. The default is `journal`
spec.helmValues.jdbcgateway.jdbc.detailsTableName	Specifies the target table for alerts.details. This field is mandatory. The default is `details`
spec.helmValues.jdbcgateway.jdbc.connections	Specifies the number of JDBC database connections to be created. This field is mandatory. The default is `1`
spec.helmValues.jdbcgateway.jdbc.maxBatchSize	Specifies the JDBC statement maximum batch row count. This field is mandatory. The default is `10`
spec.helmValues.resources.requests.cpu	The minimum required CPU core. Specify integers, fractions (e.g. 0.5), or millicore values(e.g. 100m, where 100m is equivalent to .1 core). The default is `100m`.
spec.helmValues.resources.requests.memory	The minimum memory in bytes. Specify integers with one of these suffixes: E, P, T, G, M, K, or power-of-two equivalents: Ei, Pi, Ti, Gi, Mi, Ki. The default is `128Mi`.
spec.helmValues.resources.limits.cpu	The upper limit of CPU core. Specify integers, fractions (e.g. 0.5), or millicores values(e.g. 100m, where 100m is equivalent to .1 core). The default is `500m`.
spec.helmValues.resources.limits.memory	The memory upper limit in bytes. Specify integers with suffixes: E, P, T, G, M, K, or power-of-two equivalents: Ei, Pi, Ti, Gi, Mi, Ki. The default is `512Mi`.
spec.helmValues.arch	Worker node architecture to deploy. (Supports AMD64 architecture only) Fixed to `amd64`.

Pre-installation tasks

Before installing the gateway, you must perform the following tasks:

1. Gathering ObjectServer details
2. Preparing ObjectServer communication secret
3. Configuring the database for secure connections
4. Configuring the Secrets for the gateway
5. Provisioning the database for Audit or Reporting mode
6. Creating a Persistent Volume Claim for the JDBC Driver
7. Copying the JDBC Driver into the Gateway Pod

1. Gathering ObjectServer details

For the gateway to successfully connect to ObjectServer, the gateway must be configured with the following details:

1. ObjectServer host or service details
2. ObjectServer TLS certificate if SSL protected communication is enabled.

Note: In production environments, it is recommended to use TLS/SSL enabled communications with the ObjectServer.

Use the following these steps to obtain the required details for the ObjectServer:

1. Contact your administrator to find out the ObjectServer that the JDBC Gateway should connect to. Note the NOI release name and namespace as `NOI_RELEASE_NAME` and `NOI_NAMESPACE` respectively. This information will be used when preparing the ObjectServer communication Kubernetes secret later.
2. Determine the ObjectServer NodePort service name and port number. Refer to Connecting with the ObjectServer NodePort for more info on identifying the ObjectServer NodePort and note down the service names and the NodePort number as `NOI_OBJECT_SERVER_PRIMARY_SERVICE` and `NOI_OBJECT_SERVER_PRIMARY_PORT` respectively. You may also note the backup ObjectServer service if you wish to connect to the backup ObjectServer too as `NOI_OBJECT_SERVER_BACKUP_SERVICE` and `NOI_OBJECT_SERVER_BACKUP_PORT` respectively . This information will be used when configuring the gateway.
3. Determine the TLS Proxy listening port by following the steps in Identifying the proxy listening port page and note down the TLS proxy Common Name (CN) and port number as `NOI_TLS_PROXY_CN` and `NOI_TLS_PROXY_PORT` respectively. This information will be used when configuring the gateway.
4. Determine the TLS Proxy certificate Kubernetes secret that should be used by the JDBC Gateway. This is usually set in `{{ Release.Name}}-proxy-tls-secret` secret, where `{{ Release.Name }}` is the NOI release name. Note down the secret name and namespace. This information will be used when preparing the ObjectServer communication Kubernetes secret later.
5. Get the ObjectServer user password which is required by the Gateway when creating and Insert, Delete, Update, or Control (IDUC) communication protocol connection. The credential is provided in the `{{ Release.Name }}-omni-secret`, where `{{ Release.Name }}` is the NOI release name.
6. Obtain the cluster proxy IP address.

The "Gathered Facts" table below lists the details that is gathered from the above steps. These items will be referenced in the following sections.

Item	Description and sample value
CLUSTER_MASTER_IP	The cluster master node IP address. This should be set as the `netcool.primaryIP` and optionally `netcool.backupIP` to also connect to the backup ObjectServer.

Item	Description and sample value
CLUSTER_MASTER_HOST	The cluster master node hostname. This should be set as the `netcool.primaryHost` and optionally `netcool.backupHost` to also connect to the backup ObjectServer.
NOI_RELEASE_NAME	The NOI release name. For example, `noi-m76`
NOI_NAMESPACE	Namespace where NOI is installed. For example, `default`
NOI_OBJECT_SERVER_PRIMARY_SERVICE	The primary ObjectServer Nodeport service. For example, `noi-m76-objserv-agg-primary-nodeport`. This should be set as the `netcool.primaryIDUCHost` parameter.
NOI_OBJECT_SERVER_PRIMARY_PORT	The primary ObjectServer Nodeport number. This should be set as the `netcool.primaryPort` parameter.
NOI_OBJECT_SERVER_BACKUP_SERVICE	(Optional) The backup ObjectServer Nodeport service. For example `noi-m76-objserv-agg-backup-nodeport`. This should be set as the `netcool.backupIDUCHost` parameter.
NOI_OBJECT_SERVER_BACKUP_PORT	(Optional) The backup ObjectServer Nodeport number
NOI_TLS_PROXY_CN	The NOI TLS certificate subject Common Name. For example `proxy.noi-m76.mycluster.icp`. This should be set as the `netcool.primaryHost` (and optionally `netcool.backupHost`). For more details on how to obtain the Subject Common Name, see Configuring TLS encryption with Red Hat OpenShift or Configuring TLS encryption with a custom certificate.
NOI_TLS_PROXY_PORT_1	The NOI TLS Proxy service port number (first port).
NOI_TLS_PROXY_PORT_2	(Optional) The NOI TLS Proxy service port number (second port), required to connect to backup ObjectServer.
NOI_TLS_SECRET_NAME	Secret name containing the TLS certificate of the TLS Proxy. For example `noi-m76-proxy-tls-secret`
NOI_OMNI_USER	Username for IDUC connection.
NOI_OMNI_PASSWORD	Password for IDUC connection.

2. Preparing ObjectServer communication secret

The secret can be created using the utility script ("create-noi-secret.sh") provided in the `ibm-netcool-integrations/inventory/ibmNetcoolGatewayJDBCSetup/files directory in the downloaded archive` directory. Before running this script, several pieces of information must be gathered and then configured in the script's configuration file ("create-noi-secret.config") which should be obtained following the steps in "Gathering ObjectServer Details" section above.

For this task, you will need the JDBC Gateway image in your local file system because the script requires several utility commands such as `nco_gskcmd` and `nco_aes_crypt` to add the ObjectServer certificate into the key database.

Note: If you are using a root CA signer and want to import it into the Key Database file as a trusted CA signer, see the following Technote for instructions: https://www.ibm.com/support/pages/node/1274896

1. Follow the steps in "Pulling images from IBM Entitled Registry" on page 48 to pull the JDBC Gateway image `netcool-gateway-jdbc` into your local file system. The following steps uses `netcool-gateway-jdbc:latest` as the image name and image tag for simplicity. You can list the images on the registry by following the steps in "Listing images in IBM Entitled Registry" on page 48 and select the latest `netcool-gateway-jdbc` image to pull.
2. From the command line, review and update the create-noi-secret.sh utility script configuration file (`create-noi-secret.config`) file provided in the `ibm-netcool-integrations/inventory/ibmNetcoolGatewayJDBCSetup/files directory in the downloaded archive` directory. Several items from the "Gathered Facts" table above is required when configuring the script.
3. Run the "create-noi-secret.sh" to create the Gateway-ObjectServer communication secret. The script should be run as an administrator or a user with read permissions to the NOI TLS secret so that the script can retrieve the TLS certificate file.
4. Optionally, verify that the secret is successfully created using the `kubectl describe secret <secret name> --namespace <namespace>` command.

```
kubectl describe secret jdbc-gw-noi-secret --namespace jdbcgw-ns
Name:         jdbc-gw-noi-secret
Namespace:    jdbcgw-ns
Labels:       <none>
Annotations:  <none>

Type:  Opaque

Data
====
omni.sth:             193 bytes
AuthPassword:         70 bytes
AuthUserName:         4 bytes
encryption.keyfile:   36 bytes
omni.kdb:             10088 bytes
```

3. Configuring the database for secure connections

This step is optional. It is only applicable if you are using MYSQL on Openshift and choose to use the Openshift service serving certificate to encrypt the SQL connections. If your MYSQL database is already TLS enabled, you can just extract the CA cert of your TLS cert and use when performing the Configuring the Secrets for the gateway task.

To configuring the database for secure connections, use the following steps:

1. Annotate the MySQL service to create the service certificate.

   ```
   oc annotate service mysql service.beta.openshift.io/serving-cert-secret-name=mysql-tls-secret
   ```

2. Create a ConfigMap called `mysql-ca-configmap`.

   ```
   apiVersion: v1
   kind: ConfigMap
   metadata:
     name: mysql-ca-configmap
   data:
   ```

   Annotate the `configmap` to populate it with the OpenShift Service CA certificate.

   ```
   oc annotate configmap mysql-ca-configmap service.beta.openshift.io/inject-cabundle=true
   ```

   The configmap now contains the service CA certificate.

3. Update the MySQL configuration to mount the certificates, specify the TLS version, and require secure connections for the client. This can be done using the Deployment or using a `configmap` for `/etc/my.cnf`

The example below updates the Deployment:

```
    spec:
      volumes:
        - name: mysql-volume-1
          emptyDir: {}
        - name: mysql-tls-service-cert
          secret:
            secretName: mysql-tls-secret
        - name: mysql-ca-configmap
          configMap:
            name: mysql-ca-configmap
...
          volumeMounts:
            - name: mysql-volume-1
              mountPath: /var/lib/mysql
            - name: mysql-tls-service-cert
              mountPath: /etc/mysql/ssl
            - name: mysql-ca-configmap
              mountPath: /etc/mysql/ca
...
          args:
            - '--require-secure-transport=ON'
            - '--ssl_ca=/etc/mysql/ca/service-ca.crt'
            - '--ssl_cert=/etc/mysql/ssl/tls.crt'
            - '--ssl_key=/etc/mysql/ssl/tls.key'
            - '--tls-version=TLSv1.2'
            - '--sql-mode='
```

Apply the changes and make sure that a new MySQL Pod is created and starts up successfully.

4. Configuring the Secrets for the gateway

To configure the secrets for the gateway, use the following steps:

1. Create the Secret for the Service CA Certificate

   Extract the CA certificate from the configmap created earlier into a file called `tls.crt`.

   ```
   oc describe configmap mysql-ca-configmap | sed -ne '/-BEGIN CERTIFICATE-/,/-END
   CERTIFICATE-/p' > tls.crt
   ```

   Create the Secret

   ```
   oc create secret generic jdbcgw-mysql-tls-secret --from-file=tls.crt
   ```

   Specify the Secret in the custom resource YAML.

   ```
     tlsSecretName: "jdbcgw-mysql-tls-secret"
   ```

2. Create the MySQL Credentials Secret

   ```
   oc create secret generic jdbcgw --from-literal=Username=$MYSQL_USERNAME --from-
   literal=Password=$MYSQL_PASSWORD
   ```

   Specify the Secret in the custom resource YAML.

   ```
   credentialsSecretName: "jdbcgw-mysql-cred-secret"
   ```

5. Provisioning the database for Audit or Reporting mode

The scripts can be obtained from the operator bundle package under the `inventory/ibmNetcoolGatewayJDBCSetup` directory.

```
$ tar xvf ibm-noi-int-v1.3.2.tar.gz
./
./ibm-netcool-integrations-1.3.2.tgz
./ibm-netcool-integrations-1.3.2-images.csv
$ tar zxvf ibm-netcool-integrations-1.3.2.tgz
ibm-netcool-integrations
```

```
ibm-netcool-integrations/LICENSE
ibm-netcool-integrations/README.md
ibm-netcool-integrations/case.yaml
ibm-netcool-integrations/inventory
ibm-netcool-integrations/inventory/ibmNetcoolGatewayJDBCSetup
ibm-netcool-integrations/inventory/ibmNetcoolGatewayJDBCSetup/README.md
ibm-netcool-integrations/inventory/ibmNetcoolGatewayJDBCSetup/actions.yaml
ibm-netcool-integrations/inventory/ibmNetcoolGatewayJDBCSetup/files
ibm-netcool-integrations/inventory/ibmNetcoolGatewayJDBCSetup/files/audit
ibm-netcool-integrations/inventory/ibmNetcoolGatewayJDBCSetup/files/audit/mysql
ibm-netcool-integrations/inventory/ibmNetcoolGatewayJDBCSetup/files/audit/mysql/mysql.sql
ibm-netcool-integrations/inventory/ibmNetcoolGatewayJDBCSetup/files/audit/mysql/mysqlinstall
ibm-netcool-integrations/inventory/ibmNetcoolGatewayJDBCSetup/files/reporting
ibm-netcool-integrations/inventory/ibmNetcoolGatewayJDBCSetup/files/reporting/mysql
ibm-netcool-integrations/inventory/ibmNetcoolGatewayJDBCSetup/files/reporting/mysql/
mysql_reporting.sql
ibm-netcool-integrations/inventory/ibmNetcoolGatewayJDBCSetup/files/reporting/mysql/
setup_MySql_Reporting.sh
ibm-netcool-integrations/inventory/ibmNetcoolGatewayJDBCSetup/inventory.yaml
ibm-netcool-integrations/inventory/ibmNetcoolGatewayJDBCSetup/resources.yaml
...
```

The Audit scripts are located at `ibm-netcool-integrations/inventory/ibmNetcoolGatewayJDBCSetup/files/audit` while the Reporter scripts are located at `ibm-netcool-integrations/inventory/ibmNetcoolGatewayJDBCSetup/files/reporting`.

To provision the Reporter database, use the following steps.

1. Copy the scripts to the MySQL Pod.

   ```
   export MYSQL_POD=$(oc get pods --no-headers | grep 'mysql' | grep -v 'deploy' | awk '{print
   $1}')
   oc cp setup_MySql_Reporting.sh $MYSQL_POD:/tmp
   oc cp mysql_reporting.sql $MYSQL_POD:/tmp
   ```

2. Run the scripts to provision the database.

   ```
   export MYSQL_USERNAME=netcool
   export MYSQL_PASSWORD=<password>
   export MYSQL_DATABASE=reporter
   oc exec $MYSQL_POD -- /bin/bash -c "/tmp/setup_MySql_Reporting.sh --user $MYSQL_USERNAME --
   passwd $MYSQL_PASSWORD --database $MYSQL_DATABASE --dropdb true"
   ```

Provide the necessary input when prompted by the script.

6. Creating a Persistent Volume Claim for the JDBC Driver

The gateway requires the JDBC driver for the database to be installed. The JDBC driver must be copied to a Persistent Volume (PV) storage that will be accessed by the gateway. To allow the gateway to access the PV, a Persistent Volume Claim (PVC) must be created. This can be done using one of the following methods:

1. Allow the operator to automatically create a PVC for the Deployment.

   The operator will automatically create a PVC with a storage size of 10Mi using the default storage class name on the cluster for the Deployment. PVCs that are automatically created by the operator is removed when the Deployment is deleted. To use this method, set `existingClaimName` to empty.

2. Pre-create the PVC.

   If you intend to perform many Deployments it is recommended to pre-create a PVC in order to simplify the effort of providing the JDBC Driver for each Deployment. Pre-created PVCs will not be deleted when the Deployment is deleted so you may re-use these PVCs in subsequent Deployments. To use this method, create a PVC into the namespace of your Deployment and specify the `existingClaimName` with the name of the PVC.

An example PVC `jdbc-gw-pvc.yaml` is available in the pre-install directory.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: jdbc-gw-pvc
```

```
spec:
  accessModes:
  - ReadWriteMany
  volumeMode: Filesystem
  resources:
    requests:
      storage: 10Mi
  storageClassName:
```

The example above will provision a PVC with the name `jdbc-gw-pvc` that specifies the volumeMode of `FileSystem`, accessMode of `ReadWriteMany`, 10MB of disk space from the volume and uses the default storageClass configured for the volume. As the operator provisions a Deployment, the accessMode should be defined as `ReadOnlyMany` or `ReadWriteMany`.The storage size of 10MB is to accommodate the JDBC Driver, increase the storage size if more files are to be stored in the volume.

Apply the PVC as follows:

```
oc apply -f jdbc-gw-pvc.yaml
```

Specify the PVC name in the custom resource YAML under `existingClaimName`:

```
existingClaimName: "jdbc-gw-pvc"
```

7. Copying the JDBC Driver into the Gateway Pod

The JDBC driver JAR file is not provided with the operator. You must obtain the appropriate driver from your database administrator.

Apply the following steps to copy the JDBC driver into the gateway Pod. These steps should be performed when the gateway has just been deployed and has started an initContainer called `jdbcdriver-wait`. The JDBC driver is to be copied into the Pod as instructed which will store the JDBC Driver into the PVC used by the Pod.

If you have pre-created a PVC from the previous section and have already copied the JDBC Driver into the Pod earlier, the gateway will be able to access the JDBC Driver from the pre-created PVC and startup successfully. You may skip the following steps if this is the case.

```
export POD_NAME=$(oc get pods -l app.kubernetes.io/instance=${CV_TEST_CHART_RELEASE} -l
app.kubernetes.io/component=jdbcgate -o=name | sed "s/^.\{4\}//")
export JDBC_DRIVER_DIR=/home/netcool/etc/jdbc
export INIT_CTR_NAME=jdbcdriver-wait
oc cp mysql-connector-java-8.0.26.jar $POD_NAME:$JDBC_DRIVER_DIR -c $INIT_CTR_NAME
```

Deploying the Probe for Prometheus Integration

IBM Netcool/OMNIbus Probe for Prometheus creates the webhook endpoints to receive notification in the form of HTTP POST requests from Prometheus Alert Manager.

Note: The IBM Netcool Operations Insight Event Integrations Operator must be deployed before deploying this integration. See “Installing the IBM Netcool Operations Insight Event Integrations Operator” on page 21.

To deploy a new instance of the probe, review the custom resource YAML template supplied with the operator package, update the parameters if required, and save the configurable parameters in a file, for example `probes_v1beta1_prometheusprobe.yaml`.

Then deploy the custom resource using the updated file:

```
kubectl apply -f probes_v1beta1_prometheusprobe.yaml --namespace <NAMESPACE>
```

Note: If you are installing using the CLI, you can get the custom resource Yaml file from the `ibm-netcool-integrations/inventory/netcoolIntegrationsOperatorSetup/files/op-cli/deploy/crs` directory in our CASE archive. If you are installing using the Operator Lifecycle Manager (OLM) Web console, the custom resource is provided in the YAML view.

The following table describes the configurable parameters for this probe and lists their default values.

Configurable parameters

Parameter name	Description
spec.license.accept	The license state of the image being deployed. Enter `true` to install and use the image. The default value is `false`.
spec.version	Probe application version. Accepted versions are `4.7.0`, `4.8.0` or `4.9.0`. The default value is `4.9.0`.
spec.helmValues.image.useDefaultOperandImages	Use default operand images. If `true`, the operator will use default images for the operands ignoring the image settings in the custom resource YAML file. The operator will perform an upgrade with the latest images from the operator. If `false`, the operator will use the images specified in the custom resource YAML file to deploy the operand. This should be set to `false` if you are pulling from a private repository and no image mirroring is configured in the cluster. The default is `true`.
spec.helmValues.image.repository	Probe image repository. Update this repository name to pull from a private image repository. The default value is `cp.icr.io/cp/noi-int/netcool-probe-messagebus`, and the image name must not be changed.
spec.helmValues.image.tag	The probe image tag. The default value is `25.0.0-amd64`.
spec.helmValues.image.digest	The digest to identify the probe image. If unspecified, the image tag is used. The default value is `sha256:7d524c2bebd30640bbbc2ed89367800b65b6cfdc2c53a1e5785577284af36956`.
spec.helmValues.image.testRepository	Utility image repository. Update this repository name to pull from a private image repository. The default is `cp.icr.io/cp/noi-int/netcool-integration-util`, and the image name must not be changed.
spec.helmValues.image.testImageDigest	Digest to identify the probe image. If unspecified, the image tag is used. The default value is `sha256:97beb12f8660cdc2392507e9e0d42c026b7ff41f3a5c811785621af8a3906e0a`.
spec.helmValues.image.testImageTag	Utility image tag. The default is `3.21.6-amd64`.
spec.helmValues.image.pullPolicy	The image pull policy. The default value is `IfNotPresent`.

Parameter name	Description
spec.helmValues.global.image.secretName	The name of the secret containing the docker config to pull the image from a private repository. Leave this parameter blank if the probe image already exists in the local image repository or the Service Account has a been assigned with an Image Pull Secret. The default value is `nil`.
spec.helmValues.global.image.useTag	Configures pods to pull images using tags or digests. Set this property to `false` to use digest to identify the container images. Set this property to `true` to use tags instead of digests to identify the container images. Default value is `false`.
spec.helmValues.global.serviceAccountName	Name of the service account to be used by the pods. If the Cluster Administrator has already created a service account in the namespace, specify the name of the service account here. If left blank, the probe will automatically create a new service account in the namespace when it is deployed. This new service account will be removed from the namespace when the probe is removed. The default is `nil`.
spec.helmValues.global.enableNetworkPolicy	Enables NetworkPolicy to allow traffic from and to the pod. The default is `true`.
spec.helmValues.netcool.connectionMode	The connection mode to use when connecting to the Netcool/OMNIbus ObjectServer. Refer to Securing probe and ObjectServer communications for more details. **Note:** Refer to the limitations section for more details on available connection modes for your environment. The default is `default`.
spec.helmValues.netcool.primaryServer	The primary Netcool/OMNIbus server to connect to. This is usually set to `NCOMS` or `AGG_P`. The default value is `nil`.
spec.helmValues.netcool.primaryHost	The host of the primary Netcool/OMNIbus server. Specify the ObjectServer hostname or IP address. The default value is `nil`
spec.helmValues.netcool.primaryPort	The port of the primary Netcool/OMNIbus server. The default value is `4100`.
spec.helmValues.netcool.backupServer	The backup Netcool/OMNIbus server to connect to. If the **backupServer**, **backupHost** and **backupPort** parameters are defined in addition to the **primaryServer**, **primaryHost**, and **primaryPort** parameters, the probe will be configured to connect to a virtual object server pair called \`AGG_V\`. If no backup ObjectServer is configured, only the primary server parameters will be used. The default value is `nil`.

Parameter name	Description
spec.helmValues.netcool.backupHost	The host of the backup Netcool/OMNIbus server. Specify the ObjectServer hostname or IP address. The default value is `nil`
spec.helmValues.netcool.backupPort	The port of the backup Netcool/OMNIbus server. The default value is `nil`.
spec.helmValues.netcool.secretName	This is a pre-created secret for `AuthOnly`, `SSLOnly` or `SSLAndAuth` connection mode. Certain fields are required depending on the connection modes. The default is `nil`.
spec.helmValues.probe.messageLevel	The probe log message level. The default value is `warn`.
spec.helmValues.probe.setUIDandGID	When set to true, the operator will specify the UID and GID values for the netcool user else the netcool user will not be assigned any UID or GID by the operator. Default value is `true`.
spec.helmValues.probe.sslServerCommonName	Set this parameter to a comma-separated list of acceptable SSL common names for when connecting to the ObjectServer using SSL. This should be set when the `CommonName` field of the received certificate does not match the name specified by the **`primaryServer`** property. When a **`backupServer`** is specified, the probe creates an ObjectServer pair with the name `AGG_V`. Set this parameter if the `CommonName` of the certificate does not match `AGG_V`. The default is `nil`.
spec.helmValues.probe.locale	Probe environment locale setting. Used as the LC_ALL environment variable. Default value is `en_US.utf8`
spec.helmValues.probe.enableTransportDebugLog	Set to `true` to enable common Transport Module debug logging for troubleshooting purposes. Default value is `false`.
spec.helmValues.prometheusProbe.replicaCount	The number of deployment replicas of the Prometheus Probe. This will be ignored if `prometheusProbe.autoscaling.enabled=true` and will use the **minReplicas** as the **replicaCount**. The default value is `1`.
spec.helmValues.prometheusProbe.ingress.enabled	Set this parameter to `true` to enable Ingress. The default value is `false`.
spec.helmValues.prometheusProbe.ingress.host	Sets the virtual host name for the same IP address. The value must have the Ingress controller default domain as a suffix. Default value is `nil`.
spec.helmValues.prometheusProbe.ingress.tlsEnabled	Enables the TLS Webhook for secure connectivity on the ingress route. Default value is `false`.
spec.helmValues.prometheusProbe.autoscaling.enabled	Set this parameter to `false` to disable auto-scaling. The default value is `true`.

Parameter name	Description
spec.helmValues.prometheusProbe.autoscaling.minReplicas	The minimum number of probe replicas. The default value is 1.
spec.helmValues.prometheusProbe.autoscaling.maxReplicas	The maximum number of probe replicas. The default value is 3.
spec.helmValues.prometheusProbe.autoscaling.cpuUtil	The target percentage CPU utilization. For example, enter `60` for `60%` target utilization. The default value is `60`.
spec.helmValues.prometheusProbe.poddisruptionbudget.enabled	Set this parameter to `true` to enable Pod Disruption Budget to maintain high availability during a node maintenance. Administrator role or higher is required to enable Pod Disruption Budget on clusters with Role Based Access Control. The default value is `false`.
spec.helmValues.prometheusProbe.poddisruptionbudget.minAvailable	The minimum number of available pods during node drain. This can be set to a number or a percentage, for example: 1 or 10%. **Caution:** Setting this parameter to 100%, or to the number of replicas, may block node drains entirely. The default value is 1.
spec.helmValues.resources.limits.cpu	The container CPU limit. The default value is `500m`.
spec.helmValues.resources.limits.memory	Use this parameter to specify the memory resource limits. The default value is `512Mi`
spec.helmValues.resources.limits.ephemeral-storage	Ephemeral storage limits. The default value is `500Mi`
spec.helmValues.resources.requests.cpu	Use this parameter to specify the CPU resource requests. The default value is `1000m`
spec.helmValues.resources.requests.memory	Use this parameter to specify the memory resource requests. The default value is `128Mi`
spec.helmValues.resources.requests.ephemeral-storage	Ephemeral storage requests. The default value is `100Mi`
spec.helmValues.arch	Use this parameter to specify the worker node architecture. This is fixed to amd64.

Post-installation steps for the Prometheus Probe

This section describes the post-installation steps required for the Prometheus Probe

Use the following steps to obtain the probe webhook URL and configure the Prometheus Alertmanager to add the probe webhook as a receiver.

1. The Prometheus Probe webhook can be reached via its service from within the cluster.
   For example, if the `PrometheusProbe` instance name is `example-prometheusprobe`,

then the webhook URL is `http://example-prometheusprobe-ibm-netcool-probe-prometheusprobe.<NAMESPACE>/probe/webhook/prometheus` , where <NAMESPACE> is the namespace where the probe is deployed. If ingress or route is enabled, then the probe webhook is exposed externally via a Route resource and the hostname should be the value set in the custom resource YAML file.

2. Use the steps in the following guide to apply a new configuration:

   Applying custom Alertmanager configuration

   The following sample configuration shows the probe webhook as a receiver:

```
global:
  resolve_timeout: '5m'
receivers:
- name: 'null'
- name: 'netcool_probeclus'
  webhook_configs:
  - url: 'http://yjprome-ibm-netcool-probe-prometheusprobe.yjoperator:80/probe/webhook/
prometheus'
    send_resolved: true

route:
  group_by:
  - alertname
  group_interval: 5m
  group_wait: 30s
  receiver: netcool_probeclus
  repeat_interval: 5s
  routes:
  - receiver: netcool_probeclus
    match:
      alertname: Watchdog
```

Deploying the Probe for Kafka Integration

IBM Netcool/OMNIbus Probe for Kafka Integrations processes Kafka events from a Kafka server.

Creates a Kafka consumer to subscribe messages from a Kafka topic.

Note: The IBM Netcool Operations Insight Event Integrations Operator must be deployed before deploying this integration. See "Installing the IBM Netcool Operations Insight Event Integrations Operator" on page 21.

To deploy a new instance of the probe, review the custom resource YAML template supplied with the operator package, update the parameters if required, and save the configurable parameters in a file, for example `probes_v1_kafkaprobe.yaml`.

Then deploy the custom resource using the updated file:

```
kubectl apply -f probes_v1_kafkaprobe.yaml --namespace <NAMESPACE>
```

Note: If you are installing using the CLI, you can get the custom resource Yaml file from the `ibm-netcool-integrations/inventory/netcoolIntegrationsOperatorSetup/files/op-cli/deploy/crs` directory in our CASE archive. If you are installing using the Operator Lifecycle Manager (OLM) Web console, the custom resource is provided in the YAML view.

The following table describes the configurable parameters for this probe and lists their default values.

Configurable parameters

Parameter name	Description
`spec.license.accept`	Use this parameter to specify the license state of the image being deployed. Enter `true` to install and use the image. The default value is `false`.

Parameter name	Description
`spec.version`	Use this parameter to specify the probe application version. The accepted versions are `3.4.0`, `3.5.0`, or `3.6.0`. The default value is `3.6.0`.
`spec.helmValues.global.image.secretName`	Use this parameter to specify the name of the secret containing the Docker Config to pull the image from a private repository. Leave blank if the probe image already exists in the local image repository or the Service Account has a been assigned with an Image Pull Secret. The default value is `nil`
`spec.helmValues.global.image.useTag`	Use this parameter to configure pods to pull images using tags or digests. Set to false to use digest to identify the container images. Set to true to use tags instead of digests to identify the container images. The default is `false`.
`spec.helmValues.global.serviceAccountName`	Use this parameter to specify the name of the service account to be used by the helm chart. If the Cluster Administrator has already created a service account in the namespace, specify the name of the service account here. If left blank, the chart will automatically create a new service account in the namespace when it is deployed. This new service account will be removed from the namespace when the chart is removed. The default is `nil`.
`spec.helmValues.global.enableNetworkPolicy`	Enables NetworkPolicy to allow traffic from and to the pod. The default is `true`.
`spec.helmValues.image.useDefaultOperandImages`	Use default operand images. If `true`, the operator will use default images for the operands ignoring the image settings in the custom resource YAML file. The operator will perform an upgrade with the latest images from the operator. If `false`, the operator will use the images specified in the custom resource YAML file to deploy the operand. This should be set to `false` if you are pulling from a private repository and no image mirroring is configured in the cluster. The default is `true`.

Parameter name	Description
`spec.helmValues.image.repository`	Use this parameter to specify the probe image repository. Update this repository name to pull from a private image repository. The image name should be set to `netcool-probe-messagebus` The default value is `cp.icr.io/cp/noi-int/netcool-probe-messagebus`
`spec.helmValues.image.tag`	Use this parameter to specify the `netcool-probe-messagebus` image tag. The default value is `25.0.0-amd64`
`spec.helmValues.image.digest`	Use this parameter to specify the digest to identify the probe image. If unspecified, the image tag is used. The default is `sha256:7d524c2bebd30640bbbc2ed89367800b65b6cfdc2c53a1e5785577284af36956`.
`spec.helmValues.image.testRepository`	The utility image repository. Update this repository name to pull from a private image repository. The default value is `cp.icr.io/cp/noi-int/netcool-integration-util`
`spec.helmValues.image.testImageTag`	Use this parameter to specify the utility image tag. The default value is `3.21.6-amd64`
`spec.helmValues.image.testImageDigest`	Use this parameter to specify the digest to identify the utility image. If unspecified, the image tag is used. The default is `sha256:97beb12f8660cdc2392507e9e0d42c026b7ff41f3a5c811785621af8a3906e0a`.
`image.pullPolicy`	Use this parameter to specify the image pull policy. The default value is `IfNotPresent`
`spec.helmValues.netcool.connectionMode`	Use this parameter to specify the connection mode to use when connecting to the Netcool/OMNIbus Object Server. See .Securing Probe and Object Server Communications for more details. **Note:** Refer to the limitations section for more details on available connection modes for your environment. The default is `default`.
`spec.helmValues.netcool.primaryServer`	Use this parameter to specify the primary Netcool/OMNIbus server that the probe should connect to (required). This is usually set to NCOMS or AGG_P. The default value is `nil`

Parameter name	Description
`spec.helmValues.netcool.primaryHost`	Use this parameter to specify the host of the primary Netcool/OMNIbus server (required). Specify the ObjectServer hostname or IP address. The default value is `nil`
`spec.helmValues.netcool.primaryPort`	Use this parameter to specify the port number of the primary Netcool/OMNIbus server (required). The default value is `4100`
`spec.helmValues.netcool.backupServer`	Use this parameter to specify the backup Netcool/OMNIbus server to connect to. If the **`backupServer`**, **`backupHost`** and **`backupPort`** parameters are defined, the probe will be configured to connect to a virtual object server pair called `AGG_V`. The default value is `nil`
`spec.helmValues.netcool.backupHost`	Use this parameter to specify the host of the backup Netcool/OMNIbus server. Specify the ObjectServer hostname or IP address. The default value is `nil`
`spec.helmValues.netcool.backupPort`	Use this parameter to specify the port of the backup Netcool/OMNIbus server. The default value is `4100`
`spec.helmValues.probe.messageLevel`	Use this parameter to specify the probe log message level. The default value is `warn`
`spec.helmValues.probe.setUIDandGID`	If true, the helm chart specifies the UID and GID values to be assigned to the `netcool` user in the container. Otherwise, when `false` the `netcool` user will not be assigned any UID or GID by the helm chart. Refer to the deployed PSP or SCC in the namespace. The default is `false`
`spec.helmValues.probe.heartbeatInterval`	Use this parameter to specify the frequency (in seconds) with which the probe checks the status of the host server. The default value is `10`
`spec.helmValues.probe.secretName`	Use this parameter to specify the Secret for authentication credentials for the HTTP and Kafka Transport. The default value is `nil`

Parameter name	Description
`spec.helmValues.probe.configPVC.name`	(Optional) Specifies a pre-created Persistent Volume Claim name to access custom probe files in Persistent Volume. If left unset, the probe will use the default Message Bus probe files. The default value is `nil`
`spec.helmValues.probe.configPVC.rulesFile`	(Optional) Specifies the path to the main probe rules file in the persistent volume. If left unset, the probe will use the default Message Bus probe rules files. This field will override the selection made for the `probe.rulesConfigmap` parameter. The default value is `nil`
`spec.helmValues.probe.configPVC.parserConfigFile`	(Optional) Specifies the path to the JSON parser configuration file in the persistent volume. If left unset, the probe will use the default JSON parser configuration. This field will override the selection made for the `spec.helmValues.probe.jsonParserConfig.messagePayload`, `spec.helmValues.probe.jsonParserConfig.messageHeader`, `spec.helmValues.probe.jsonParserConfig.jsonNestedPayload`, `spec.helmValues.probe.jsonParserConfig.jsonNestedHeader` and `spec.helmValues.probe.jsonParserConfig.messageDepth` parameters. The default value is `nil`
`spec.helmValues.probe.rulesConfigmap`	Use this parameter to specify a customized ConfigMap to be used for the rules files. This field may be left blank if not required. The default value is `nil`
`spec.helmValues.probe.jsonParserConfig.messagePayload`	Use this parameter to specify the JSON tree to be identified as message payload from the notification (kafka consumer) channel. See example for more details on how to configure the Probe's JSON parser. The default value is `json`
`spec.helmValues.probe.jsonParserConfig. messageHeader`	Use this parameter to specify a JSON tree to be identified as message header from the notification (kafka consumer) channel. Attributes from the headers will be added to the generated event. The default value is `nil`

Parameter name	Description
spec.helmValues.probe.jsonParserConfig. jsonNestedPayload	Use this parameter to specify a JSON tree within a nested JSON or JSON string to be identified as message payload from the notification (kafka consumer) channel. **messagebus.probe.jsonParserConfig.messagePayload** must be set to point to the attribute containing the JSON String. The default value is `nil`
spec.helmValues.probe.jsonParserConfig. jsonNestedHeader	Use this parameter to specify a JSON tree within a nested JSON or JSON string to be identified as message header from the notification (kafka consumer) channel. **messagebus.probe.jsonParserConfig.messageHeader** must be set to point to the attribute containing the JSON String. The default value is `nil`
spec.helmValues.probe.jsonParserConfig.messageDepth	Use this parameter to specify the number of JSON child node levels in the message to traverse during parsing. The default value is 3
spec.helmValues.probe.enableTransportDebugLog	Set to `true` to enable common Transport Module debug logging for troubleshooting purposes. Default value is `false`. The default value is `false`
spec.helmValues.probe.selfMonitoring.discardHeartbeatEvent	Set this value to `true` if you want to discard the ProbeWatch Heartbeat event. Set this value to `false` if you want to forward the ProbeWatch Heartbeat event to the ObjectServer. The default is `false`.
spec.helmValues.probe.selfMonitoring.populateMasterProbeStat	Set this value to `true` to enable a new probe metrics event to be generated by using the genevent function, which is defined within the case "Heartbeat ..." statement. The event data consists a set of OplStats probe metrics, which are forwarded to the `master.probestats` table that was created when you ran the `probestats.sql` script. Set this value to `false` if you do not want to generate this event for insertion into the `master.probestats` table. The default is `false`.
spec.helmValues.probe.selfMonitoring.logProbeStat	Set this value to `true` if you want to record the OplStats metrics in the probe log file. Details are logged at the INFO level. The metric details that are logged are defined in the case "Heartbeat ..." statement. Set this value to `false` if you do not want to record the metrics in the log file. The default is `false`.

Parameter name	Description
spec.helmValues.probe.selfMonitoring.generateThresholdEvents	Set this value to `true` if you want to generate threshold events that indicate when a particular probe metric violates a defined threshold. By default, no code is provided for threshold events within this rules file because individual preferences can vary widely. If you require threshold events, you must first decide which thresholds you want to monitor. Then, within the case "Heartbeat ..." statement, provide the code for generating threshold events. The default is `false`.
spec.helmValues.probe.configs	A multiline string parameter to configure additional probe properties. Each entry must not contain space. The default is `ProbeWatchHeartbeatInterval:0 EventLoadProfiling:'true'`.
`spec.helmValues.probe.kafka.connection. zookeeperClient.target`	Use this parameter to specify the ZooKeeper endpoint. The default value is `nil`
`spec.helmValues.probe.kafka.connection. zookeeperClient.topicWatch`	Use this parameter to enable the ZooKeeper topic watch service. The default value is `false`
`spec.helmValues.probe.kafka.connection. zookeeperClient.brokerWatch`	Use this parameter to enable the ZooKeeper broker watch service. The default value is `false`
`spec.helmValues.probe.kafka.connection.brokers`	Use this parameter to specify the broker endpoints in a comma-separated list, for example: `PLAINTEXT://kafkaserver1:9092,PLAINTEXT://kafkaserver2:9092` The default value is `nil`
`spec.helmValues.probe.kafka.connection.topics`	Use this parameter to specify the topics in a comma-separated list, for example `topicABC,topicXYZ`. The default value is `nil`
`spec.helmValues.probe.kafka.client.securityProtocol`	Use this parameter to specify the security protocol. The default value is `nil`
`spec.helmValues.probe.kafka.client.ssl. trustStoreSecretName`	Use this parameter to specify the truststore Secret name and its password. The default value is `nil`

Parameter name	Description
spec.helmValues.probe.kafka.client.ssl. keyStoreSecretName	Use this parameter to specify the keystore Secret name and its password. The default value is `nil`
spec.helmValues.probe.kafka.client. saslPlainMechanism	Use this parameter to enable the PLAIN mechanism for Simple Authentication and Security Layer connections. The default value is `false`
spec.helmValues.probe.kafka.client.sasl.jaasConfigSecretName	Use this parameter to specify the secret name which contains the Kafka consumer JAAS configuration file. The default is `nil`.
spec.helmValues.probe.kafka.client.consumer.groupId	Use this parameter to specify the group identifier. The default value is `nil`
spec.helmValues.probe.kafka.client.consumer.configs	Use this parameter to specify a multi-line string parameter to configure additional Kafka Consumer Configurations. Each entry must not contain space. The default is `key.deserializer=org.apache.kafka.common.serialization.StringDeserializer value.deserializer=org.apache.kafka.common.serialization.StringDeserializer`.
spec.helmValues.probe.replicaCount	Use this parameter to specify the number of deployment replicas. Omitted when **autoscaling.enabled** set to `true`. The default is `1`.
spec.helmValues.probe.autoscaling.enabled	Use this parameter to enable or disable auto-scaling. The default value is `true`
spec.helmValues.probe.autoscaling.minReplicas	Use this parameter to specify the minimum number of probe replicas. The default value is `1`
spec.helmValues.probe.autoscaling.maxReplicas	Use this parameter to specify the maximum number of probe replicas. The default value is 5
spec.helmValues.probe.autoscaling.cpuUtil	Use this parameter to specify the target CPU utilization. For example, enter `60` for `60%` target utilization. The default value is `60`

Parameter name	Description
`spec.helmValues.probe.poddisruptionbudget.enabled`	Use this parameter to enable or disable Pod Disruption Budget to maintain high availability during a node maintenance. The default value is `false`
`spec.helmValues.probe.poddisruptionbudget.minAvailable`	Use this parameter to specify the number of minimum available number of pods during node drain. Can be set to a number or percentage, for example, 1 or 10%. ⚠ **CAUTION:** Setting to 100% or equal to the number of replicas may block node drains entirely. The default value is `1`
`spec.helmValues.resources.limits.cpu`	Use this parameter to specify the CPU resource limits. The default value is `500m`
`spec.helmValues.resources.limits.memory`	Use this parameter to specify the memory resource limits. The default value is `512Mi`
`spec.helmValues.resources.limits.ephemeral-storage`	Ephemeral storage limits. The default value is `500Mi`
`spec.helmValues.resources.requests.cpu`	Use this parameter to specify the CPU resource requests. The default value is `250m`
`spec.helmValues.resources.requests.memory`	Use this parameter to specify the memory resource requests. The default value is `256Mi`
`spec.helmValues.resources.requests.ephemeral-storage`	Ephemeral storage requests. The default value is `100Mi`
`spec.helmValues.arch`	Use this parameter to specify the worker node architecture. This is fixed to `amd64`.

Integrating with IBM Event Streams for Red Hat Openshift

The following sections describe how to integrate with the IBM Event Streams for IBM Cloud for Red Hat OCP:

- "Connecting to the Event Streams operator from within the same cluster" on page 88
- "Connecting to the Event Streams operator from an external cluster" on page 91

Connecting to the Event Streams operator from within the same cluster

1. Get the truststore and keystore from Event Streams to be completed in the probe settings.

2. Login to the Event Streams UI. Go to the topics that you have created. Click on **Connect to this topic**. Click on **Internal**.

   - Copy down the broker details, for example:

     ```
     prod-3-brokers-kafka-bootstrap.ibm-event-stream.svc:9093
     ```

3. For truststore, download the PKCS12 and copy down the certificate password.

   - Convert the Event Stream p12 truststore file into a Java Keystore format using Java keytool:

   ```
   /opt/IBM/tivoli/netcool/platform/linux2x86/jre64_1.8.0/jre/bin/keytool -importkeystore \
   -srckeystore es-cert.p12 \
   -srcstoretype PKCS12 \
   -destkeystore client-truststore.jks  \
   -deststoretype JKS \
   -srcstorepass <certpassword> \
   -deststorepass <certpassword> \
   -noprompt
   ```

4. For keystore, click on **Connect to this topic**. Click on **Generate TLS Credentials**' (user details to connect internally). Choose all the default values. Download the zip file.

   - Convert the keystore to Java Keystore format using Java keytool:

   ```
   /opt/IBM/tivoli/netcool/platform/linux2x86/jre64_1.8.0/jre/bin/keytool -importkeystore \
   -srckeystore user.p12 \
   -srcstoretype PKCS12 \
   -destkeystore client-keystore.jks  \
   -deststoretype JKS \
   -srcstorepass <userpassword> \
   -deststorepass <userpassword> \
   -noprompt
   ```

5. Generate the secret to be completed in the probe CR file for installation:

   ```
   kubectl create secret generic -n noi-integrations kafka-es-keystore --from-file=client-
   keystore.jks --from-literal=keystore_password='userpassword'
   kubectl create secret generic -n noi-integrations kafka-es-truststore --from-file=client-
   truststore.jks --from-literal=truststore_password='certpassword'
   ```

6. When configuring the Kafka Probe CR YAML, set the `keyStoreSecretName` and `trustStoreSecretName` with the keystore secret name and truststore secret name from the previous step.

   The following code is from an example Kafka Probe CR YAML file:

   Note: Do not copy the whole YAML file, only refer to the values to be used.

   ```
   apiVersion: probes.integrations.noi.ibm.com/v1
   kind: KafkaProbe
   metadata:
     name: yjkafka
     namespace: noi-integrations
     labels:
       app.kubernetes.io/instance: yjkafka
       app.kubernetes.io/managed-by: netcool-integrations-operator
       app.kubernetes.io/name: kafkaprobe
   spec:
     helmValues:
       arch: amd64
       global:
         image:
           secretName: ''
           useTag: false
         serviceAccountName: ''
       image:
         digest: 'sha256:c0ab13fcc6bc1cb8fb85751b93e29e8d90662795deb7916dc733232ca2ab68fa'
         pullPolicy: Always
         repository: cp.icr.io/cp/noi-int/netcool-probe-messagebus
         tag: 14.0.0-amd64
         testImageDigest:
   'sha256:af9b0cf8ade76f2c32f48c54ce6ad8ad9e10a76f8af3940b5a18ff24b419f28c'
         testImageTag: 3.4.0-amd64
         testRepository: cp.icr.io/cp/noi-int/netcool-integration-util
   ```

```
    netcool:
      backupHost: ''
      backupPort: 0
      backupServer: ''
      connectionMode: default
      primaryHost: sim1.fyre.ibm.com
      primaryPort: 5008
      primaryServer: NOSSL
      secretName: ''
    probe:
      heartbeatInterval: 10
      rulesConfigmap: ''
      sslServerCommonName: ''
      kafka:
        client:
          consumer:
            configs: >-
              key.deserializer=org.apache.kafka.common.serialization.StringDeserializer
              value.deserializer=org.apache.kafka.common.serialization.StringDeserializer
            groupId: yjkafka-starter-consumer
          saslPlainMechanism: false
          securityProtocol: SSL
          ssl:
            keyStoreSecretName: kafka-es-keystore
            trustStoreSecretName: kafka-es-truststore
        connection:
          brokers: 'prod-3-brokers-kafka-bootstrap.ibm-event-stream.svc:9093'
          topics: topic1
          zookeeperClient:
            brokerWatch: false
            target: ''
            topicWatch: false
      jsonParserConfig:
        jsonNestedHeader: ''
        jsonNestedPayload: ''
        messageDepth: 3
        messageHeader: ''
        messagePayload: json.netcoolField
      replicaCount: 1
      autoscaling:
        cpuUtil: 60
        enabled: true
        maxReplicas: 5
        minReplicas: 1
      setUIDandGID: true
      locale: en_US.utf8
      secretName: ''
      poddisruptionbudget:
        enabled: false
        minAvailable: 1
      messageLevel: debug
    resources:
      limits:
        cpu: 500m
        memory: 512Mi
      requests:
        cpu: 250m
        memory: 256Mi
  license:
    accept: true
  version: 3.1.0
```

7. When probe is up and running, send events from the starter application. Verify that the probe receives the events and sends them to the event list. Check the events list for the events.

   Sample event to be sent from the starter application:

```
{ "netcoolField":{ "Identifier" : "\"TESTTESTTESTTEST\"" , "Node" : "\"Moscow\"" ,
"NodeAlias" : "Moscow" , "Manager" : "Simnet Probe", "Agent" : "\"MachineMon\"",
"AlertGroup" : "\"Systemssss\"", "AlertKey" : "", "Severity" : 4, "Summary" :
"\"TESTTESTTESTTESTTEST_JSON 早上好\"", "StateChange" : "2010-11-02T00:53:33",
"FirstOccurrence" : "2010-11-02T00:18:56", "LastOccurrence" : "2010-11-02T00:53:33",
"InternalLast" : "2010-11-02T00:53:33", "Poll" : 0, "Type" : 0, "Tally" : 15, "Class" :
3300, "Grade" : 0, "Location" : "", "OwnerUID" : 65534, "OwnerGID" : 0, "Acknowledged" :
0, "Flash" : 0, "EventId" : 12345, "ExpireTime" : 0, "ProcessReq" : 0, "SuppressEscl" :
0, "Customer" : "testing", "Service" : "", "PhysicalSlot" : 0, "PhysicalPort" : 0,
"PhysicalCard" : 0, "TaskList" : 0, "NmosSerial" : "", "NmosObjInst" : "", "NmosCauseType" :
0, "LocalNodeAlias" : 0, "LocalPriObj" : "", "LocalSecObj" : "", "LocalRootObj" : "",
"RemoteNodeAlias" : "", "RemotePriObj" : "", "RemoteSecObj" : "", "RemoteRootObj" : "",
"X733EventType" : 0, "X733ProbableCause" : 0, "X733SpecificProb" : "", "X733CorrNotif" : 0,
"URL" : "", "ExtendedAttr" : "", "ServerName" : "BOLYJ1", "ServerSerial" : 74 } }
```

8. The following sample log shows a successful connection:

```
2021-03-05T06:48:30: Debug: D-JPR-000-000: [KafkaConsumerRunnable] - Kafka Consumer Thread
Created As CONSUMER - Completed
2021-03-05T06:48:30: Debug: D-JPR-000-000: Kafka Transport Initializing Kafka Component for
CONSUMER - Successful
2021-03-05T06:48:30: Debug: D-JPR-000-000:
***********************************************************
2021-03-05T06:48:30: Debug: D-JPR-000-000: **************************** CONNECT
**********************
2021-03-05T06:48:30: Debug: D-JPR-000-000:
***********************************************************
2021-03-05T06:48:30: Debug: D-JPR-000-000: com.ibm.tivoli.oidk.ProbeImpl.connect EXITING
2021-03-05T06:48:30: Debug: D-JPR-000-000: com.ibm.tivoli.oidk.ProbeImpl.subscribe ENTERING
2021-03-05T06:48:30: Debug: D-JPR-000-000:
***********************************************************
2021-03-05T06:48:30: Debug: D-JPR-000-000: **************************** SUBCRIBE
*********************
2021-03-05T06:48:30: Debug: D-JPR-000-000:
***********************************************************
2021-03-05T06:48:30: Debug: D-JPR-000-000: Kafka transport subscribing component for
CONSUMER - Started
2021-03-05T06:48:30: Debug: D-JPR-000-000: [KafkaConsumerRunnable] - Kafka Consumer Thread
Subscribing to Topic(s) - Started
2021-03-05T06:48:30: Debug: D-JPR-000-000: [KafkaConsumerRunnable] - Kafka Consumer Thread
Subscribing to following Topics = [topic1]
2021-03-05T06:48:30: Debug: D-JPR-000-000: [KafkaConsumerRunnable] - Kafka Consumer Thread
Subscribing to Topic(s) - Completed
2021-03-05T06:48:30: Debug: D-JPR-000-000: Kafka transport subscribing component for
CONSUMER - Successful
2021-03-05T06:48:30: Debug: D-JPR-000-000:
***********************************************************
2021-03-05T06:48:30: Debug: D-JPR-000-000: *********************** GET ACTIVE ALARMS
*****************
2021-03-05T06:48:30: Debug: D-JPR-000-000:
***********************************************************
2021-03-05T06:48:30: Debug: D-JPR-000-000: Kafka transport runs consumer thread - Starting
2021-03-05T06:48:30: Debug: D-JPR-000-000: Kafka transport runs consumer thread - Started
Successfully
2021-03-05T06:48:30: Debug: D-JPR-000-000: [KafkaConsumerRunnable] - Kafka consumer thread
starts.
2021-03-05T06:48:30: Debug: D-JPR-000-000: com.ibm.tivoli.oidk.ProbeImpl.subscribe EXITING
2021-03-05T06:48:30: Debug: D-JPR-000-000:
com.ibm.tivoli.netcool.omnibus.probe.framework.ProbeRunner.connect EXITING
2021-03-05T06:48:30: Debug: D-JPR-000-000:
com.ibm.tivoli.netcool.omnibus.probe.framework.ProbeRunner.resetRetry ENTERING
2021-03-05T06:48:30: Debug: D-JPR-000-000:
com.ibm.tivoli.netcool.omnibus.probe.framework.ProbeRunner.resetRetry EXITING
2021-03-05T06:48:30: Debug: D-JPR-000-000: [KafkaClientTimer] - Waiting for first poll to
succeed
2021-03-05T06:48:30: Debug: D-JPR-000-000:
com.ibm.tivoli.netcool.omnibus.probe.framework.ResyncManager.schedule ENTERING
2021-03-05T06:48:30: Debug: D-JPR-000-000:
com.ibm.tivoli.netcool.omnibus.probe.framework.ResyncManager.schedule EXITING
2021-03-05T06:48:30: Information: I-JPR-000-000: Probe connected
```

Connecting to the Event Streams operator from an external cluster

1. Login to the Event Streams UI. Go the topics you have created. Click on **Connect to this topic**. Click on **External**. Copy down the broker details. For example:

```
prod-3-brokers-kafka-bootstrap-ibm-event-stream.apps.sqaocp4520.cp.fyre.ibm.com:443
```

2. For truststore, download the PKCS12 keystore containing the Event Stream server certificate and copy down the certificate password. Click on **Generate SCRAM credentials** (user details to connect externally). Choose all the default values. Download the zip files (you can only download when connecting to internal).

- Convert the keystore to Java Keystore format using Java keytool:

```
/opt/IBM/tivoli/netcool/platform/linux2x86/jre64_1.8.0/jre/bin/keytool -importkeystore \
-srckeystore es-cert.p12 \
-srcstoretype PKCS12 \
-destkeystore client-truststore.jks  \
-deststoretype JKS \
-srcstorepass <certpassword> \
-deststorepass <certpassword> \
-noprompt
```

3. Generate the secret to be completed in the probe CR file for installation:

```
kubectl create secret generic -n noi-integrations kafka-es-truststore --from-file=client-
truststore.jks --from-literal=truststore_password='certpassword'
```

4. From the user details in the downloaded in Step 3, create secret for JAAS user configuration and set the secret name as the `spec.helmValues.probe.kafka.client.consumer.sasl.jaasConfigSecretName` parameter value..

   - Create the `jaas-config-file.txt`:

```
KafkaClient {
org.apache.kafka.common.security.scram.ScramLoginModule required username="externaluser"
password="<userpassword>";
};
```

   - Create the `secret`:

```
kubectl create secret generic my-jaas-config -n noi-integrations --from-file=kafka-client-
jaas=jaas-config-file.txt
```

5. Set the SASL mechanism to `SCRAM-SHA-512` in the `spec.helmValues.probe.kafka.client.consumer.configs` parameter as shown in the example below. Refer to the sample probe CR YAML below to check where this value is placed.
6. Install the probe instance using all the values above. The following code is from an example Kafka Probe CR YAML file:

   Note: Do not copy the whole YAML file, only refer to the values to be used.

```
apiVersion: probes.integrations.noi.ibm.com/v1
kind: KafkaProbe
metadata:
  name: yjkafka
  labels:
    app.kubernetes.io/instance: yjkafka
    app.kubernetes.io/managed-by: netcool-integrations-operator
    app.kubernetes.io/name: kafkaprobe
  namespace: noi-integrations
spec:
  helmValues:
    arch: amd64
    global:
      image:
        secretName: ''
        useTag: false
      serviceAccountName: ''
    image:
      digest: 'sha256:c0ab13fcc6bc1cb8fb85751b93e29e8d90662795deb7916dc733232ca2ab68fa'
      pullPolicy: Always
      repository: cp.icr.io/cp/noi-int/netcool-probe-messagebus
      tag: 14.0.0-amd64
      testImageDigest:
'sha256:af9b0cf8ade76f2c32f48c54ce6ad8ad9e10a76f8af3940b5a18ff24b419f28c'
      testImageTag: 3.4.0-amd64
      testRepository: cp.icr.io/cp/noi-int/netcool-integration-util
    netcool:
      backupHost: ''
      backupPort: 0
      backupServer: ''
      connectionMode: SSLAndAuth
      primaryHost: '10.17.104.243'
      primaryPort: 32278
      primaryServer: 'AGG_P'
      secretName: 'noi255secret'
    probe:
      heartbeatInterval: 10
      rulesConfigmap: ''
      sslServerCommonName: 'noi255-proxy.pink.svc'
      kafka:
        client:
          consumer:
            configs: >-
              key.deserializer=org.apache.kafka.common.serialization.StringDeserializer
              value.deserializer=org.apache.kafka.common.serialization.StringDeserializer
              sasl.mechanism=SCRAM-SHA-512
            groupId: 'yjkafka-starter-consumer'
          sasl:
            jaasConfigSecretName: 'my-jaas-config'
          saslPlainMechanism: false
          securityProtocol: 'SASL_SSL'
          ssl:
            keyStoreSecretName: ''
            trustStoreSecretName: 'kafka-es-truststore'
        connection:
          brokers: 'prod-3-brokers-kafka-bootstrap-ibm-event-
stream.apps.sqaocp4520.cp.fyre.ibm.com:443'
          topics: 'topic1'
          zookeeperClient:
            brokerWatch: false
            target: ''
            topicWatch: false
      jsonParserConfig:
        jsonNestedHeader: ''
        jsonNestedPayload: ''
        messageDepth: 3
        messageHeader: ''
        messagePayload: json.netcoolField
      replicaCount: 1
      autoscaling:
        cpuUtil: 60
        enabled: true
        maxReplicas: 5
        minReplicas: 1
      setUIDandGID: true
      locale: en_US.utf8
      secretName: ''
      poddisruptionbudget:
        enabled: false
        minAvailable: 1
      messageLevel: debug
    resources:
      limits:
        cpu: 500m
        memory: 512Mi
      requests:
        cpu: 250m
        memory: 256Mi
  license:
    accept: true
  version: 3.1.0
```

7. When the probe is up and running, send events from the starter application. Verify that the probe receives the events and sends to event list. Check the events list for the events.

   Sample event to be sent from the starter application:

```
{ "netcoolField":{ "Identifier" : "\"TESTTESTTESTTEST\"" , "Node" : "\"Moscow\"" ,
"NodeAlias" : "Moscow" , "Manager" : "Simnet Probe", "Agent" : "\"MachineMon\"",
"AlertGroup" : "\"Systemssss\"", "AlertKey" : "", "Severity" : 4, "Summary" :
"\"TESTTESTTESTTESTTEST_JSON 早上好\"", "StateChange" : "2010-11-02T00:53:33",
"FirstOccurrence" : "2010-11-02T00:18:56", "LastOccurrence" : "2010-11-02T00:53:33",
"InternalLast" : "2010-11-02T00:53:33", "Poll" : 0, "Type" : 0, "Tally" : 15, "Class" :
3300, "Grade" : 0, "Location" : "", "OwnerUID" : 65534, "OwnerGID" : 0, "Acknowledged" :
0, "Flash" : 0, "EventId" : 12345, "ExpireTime" : 0, "ProcessReq" : 0, "SuppressEscl" :
0, "Customer" : "testing", "Service" : "", "PhysicalSlot" : 0, "PhysicalPort" : 0,
"PhysicalCard" : 0, "TaskList" : 0, "NmosSerial" : "", "NmosObjInst" : "", "NmosCauseType" :
0, "LocalNodeAlias" : 0, "LocalPriObj" : "", "LocalSecObj" : "", "LocalRootObj" : "",
"RemoteNodeAlias" : "", "RemotePriObj" : "", "RemoteSecObj" : "", "RemoteRootObj" : "",
"X733EventType" : 0, "X733ProbableCause" : 0, "X733SpecificProb" : "", "X733CorrNotif" : 0,
"URL" : "", "ExtendedAttr" : "", "ServerName" : "BOLYJ1", "ServerSerial" : 74 } }
```

8. The following sample log shows a successful connection:

```
2021-03-05T06:48:30: Debug: D-JPR-000-000: [KafkaConsumerRunnable] - Kafka Consumer Thread
Created As CONSUMER - Completed
2021-03-05T06:48:30: Debug: D-JPR-000-000: Kafka Transport Initializing Kafka Component for
CONSUMER - Successful
2021-03-05T06:48:30: Debug: D-JPR-000-000:
***********************************************************
2021-03-05T06:48:30: Debug: D-JPR-000-000: **************************** CONNECT
**********************
2021-03-05T06:48:30: Debug: D-JPR-000-000:
***********************************************************
2021-03-05T06:48:30: Debug: D-JPR-000-000: com.ibm.tivoli.oidk.ProbeImpl.connect EXITING
2021-03-05T06:48:30: Debug: D-JPR-000-000: com.ibm.tivoli.oidk.ProbeImpl.subscribe ENTERING
2021-03-05T06:48:30: Debug: D-JPR-000-000:
***********************************************************
2021-03-05T06:48:30: Debug: D-JPR-000-000: **************************** SUBCRIBE
*********************
2021-03-05T06:48:30: Debug: D-JPR-000-000:
***********************************************************
2021-03-05T06:48:30: Debug: D-JPR-000-000: Kafka transport subscribing component for
CONSUMER - Started
2021-03-05T06:48:30: Debug: D-JPR-000-000: [KafkaConsumerRunnable] - Kafka Consumer Thread
Subscribing to Topic(s) - Started
2021-03-05T06:48:30: Debug: D-JPR-000-000: [KafkaConsumerRunnable] - Kafka Consumer Thread
Subscribing to following Topics = [topic1]
2021-03-05T06:48:30: Debug: D-JPR-000-000: [KafkaConsumerRunnable] - Kafka Consumer Thread
Subscribing to Topic(s) - Completed
2021-03-05T06:48:30: Debug: D-JPR-000-000: Kafka transport subscribing component for
CONSUMER - Successful
2021-03-05T06:48:30: Debug: D-JPR-000-000:
***********************************************************
2021-03-05T06:48:30: Debug: D-JPR-000-000: *********************** GET ACTIVE ALARMS
*****************
2021-03-05T06:48:30: Debug: D-JPR-000-000:
***********************************************************
2021-03-05T06:48:30: Debug: D-JPR-000-000: Kafka transport runs consumer thread - Starting
2021-03-05T06:48:30: Debug: D-JPR-000-000: Kafka transport runs consumer thread - Started
Successfully
2021-03-05T06:48:30: Debug: D-JPR-000-000: [KafkaConsumerRunnable] - Kafka consumer thread
starts.
2021-03-05T06:48:30: Debug: D-JPR-000-000: com.ibm.tivoli.oidk.ProbeImpl.subscribe EXITING
2021-03-05T06:48:30: Debug: D-JPR-000-000:
com.ibm.tivoli.netcool.omnibus.probe.framework.ProbeRunner.connect EXITING
2021-03-05T06:48:30: Debug: D-JPR-000-000:
com.ibm.tivoli.netcool.omnibus.probe.framework.ProbeRunner.resetRetry ENTERING
2021-03-05T06:48:30: Debug: D-JPR-000-000:
com.ibm.tivoli.netcool.omnibus.probe.framework.ProbeRunner.resetRetry EXITING
2021-03-05T06:48:30: Debug: D-JPR-000-000: [KafkaClientTimer] - Waiting for first poll to
succeed
2021-03-05T06:48:30: Debug: D-JPR-000-000:
com.ibm.tivoli.netcool.omnibus.probe.framework.ResyncManager.schedule ENTERING
2021-03-05T06:48:30: Debug: D-JPR-000-000:
com.ibm.tivoli.netcool.omnibus.probe.framework.ResyncManager.schedule EXITING
2021-03-05T06:48:30: Information: I-JPR-000-000: Probe connected
```

Integrating with IBM Event Streams for IBM Cloud

The following sections describe how to integrate the Probe for Kafka with IBM Event Streams for IBM Cloud:

- "IBM Event Streams for IBM Cloud Setup" on page 95
- "Installing the Kafka instance using OLM UI" on page 95
- "Installing the Kafka instance using cloudctl case commands" on page 98
- "Testing with IBM Event Streams for IBM Cloud" on page 98

IBM Event Streams for IBM Cloud Setup

1. To start creating IBM Event Streams for IBM Cloud, see https://cloud.ibm.com/docs/EventStreams?topic=EventStreams-getting_started .

   Note: IBM Event Streams for IBM Cloud only supports `SASL_SSL`. It does not support `PLAINTEXT`.
2. When you download, the sample producer/consumer should be the standard configuration as shown below. Check for any changes in the configuration files downloaded from the producer/consumer. The details of the broker, username and token can be retrieved from the UI:

   ```
   topic : kafka-java-console-sample-topic
   consumer.groupid : test-consumer-group
   ```
3. The secret shown below is created for the connection of the probe to the IBM Event Streams for IBM Cloud. The details of the username/token were retrieved in Step 2.

   ```
   kubectl create secret generic -n noi-integrations cloudes
   --from-literal=http_username='token' --from-literal=http_password='apikey' --from-
   literal=kafka_username='token' --from-literal=kafka_password='apikey'
   ```
4. When you start the sample producer, the producer will start sending a sample event. In order to change the events so that probe will receive events that contain more information, make the following change:

   a. Edit `event-streams-samples/kafka-java-console-sample/src/main/java/com/eventstreams/samples/ProducerRunnable.java`

   b. Add the following string below at `String Message`, comment the existing `String Message` and save the file.

   ```
   String message = "{\"response\":{\"Manager\":\"Kafka\",\"Node\":\"Kafka SASL_SSL
   Test\",\"Agent\":\"cloudkafka\",\"AlertGroup\":\"pod
   status\",\"AlertKey\":\"111\",\"Severity\":\"3\",\"Summary\":\"for on perm testing actual
   dove
   4\",\"status\":0,\"startRow\":0,\"endRow\":0,\"totalRows\":1,\"data\":1535118829745,\"erro
   rs\":null}}";
   ```

   c. Run `gradle clean && gradle build` again for the changes to take effect.

   d. Start the producer and the consumer should start receiving the new events.

Installing the Kafka instance using OLM UI

Note: For the steps in this section, the sample yaml file is for the connection with IBM Event Streams for IBM Cloud.

1. Install Catalog:

   a. Go to **OCP Console**.

   b. Go to **Administration > Cluster Settings > Global Configuration > OperatorHub > Sources > Create Catalog Source**.

   A sample Catalog Source resource is shown below. If you would like to specify the image using its digest, the digest can be retrieved from the `images.csv` file from CASE bundle.

   Once Catalog is installed successfully, the number of operators will be changed to 1.

   ```
   apiVersion: operators.coreos.com/v1alpha1
   kind: CatalogSource
   metadata:
     name: ibm-netcool-integrations-operator-catalog
     namespace: REPLACE_NAMESPACE
   spec:
     displayName: IBM Netcool Operations Insight Event Integrations Catalog
     publisher: IBM
     sourceType: grpc
     image: docker.io/ibmcom/ibm-netcool-integrations-operator-catalog:latest
     updateStrategy:
   ```

```
    registryPoll:
      interval: 45m
```

2. Install the operator:

   a. Go to **Operators > OperatorHub**. Search for `IBM Netcool Operations Insight Event Integrations Operator`. Click **Install**. Choose the namespace. Choose **Auto** or **Manual**.

   b. Go to **Operators > Installed Operators**. If the installation of operator is successful, the **Status** will be shown as `Succeeded`. Alternatively you can check using command line on the status of the pods:

   ```
   oc get pods -n <namespace>
   ```

   If the operator is not starting up, check the logs and solve the issue:

   ```
   oc logs po/<podname> -n namespace
   ```

3. Install the probe instance:

   a. Go to **Operators > Installed Operator**. Click on "IBM Netcool Operations Insight Event Integrations Operator", Create Instance at KafkaProbe.

   b. Update the yaml files properties as below:

   ```
   probe.kafka.client.saslPlainMechanism: true
   probe.kafka.client.securityProtocol: SASL_SSL
   messagePayload: json.response
   ```

   For these three properties, information can be retrieved from IBM Event Streams for IBM Cloud sample producers/consumers downloaded.

   ```
   probe.kafka.client.consumer.groupid
   probe.kafka.connection.brokers
   probe.kafka.connection.topic
   ```

   Retrieve the `username` and `apikey` from IBM Event Streams for IBM Cloud following the procedures in https://cloud.ibm.com/docs/EventStreams.

   Then create a secret for the probe, for example:

   ```
       kubectl create secret generic cloudes \
       --namespace noi-integrations \
       --from-literal=http_username='<username>' \
       --from-literal=http_password='<apikey>'  \
       --from-literal=kafka_username='<username>' \
       --from-literal=kafka_password='<apikey>'
   ```

   Where `<username>` and `<apikey>` are the username and API token retrieved from IBM Event Streams for IBM Cloud.

   ```
       probe:
         secretName: cloudes
   ```

   Below is the sample yaml file:

   ```
   apiVersion: probes.integrations.noi.ibm.com/v1
   kind: KafkaProbe
   metadata:
    name: example-kafkaprobe
    labels:
      app.kubernetes.io/instance: example-kafkaprobe
      app.kubernetes.io/managed-by: netcool-integrations-operator
      app.kubernetes.io/name: kafkaprobe
    namespace: yjoperator
   spec:
    helmValues:
      arch: amd64
      global:
        image:
   ```

```
        secretName: ''
        useTag: false
      serviceAccountName: ''
    image:
      digest: 'sha256:db93b631dde5f724e0234842f56235893f1697e351e3f7e8b1e044905cbdcd07'
      pullPolicy: Always
      repository: cp.icr.io/cp/noi-int/netcool-probe-messagebus
      tag: 13.2.0-amd64
      testImageDigest:
'sha256:af9b0cf8ade76f2c32f48c54ce6ad8ad9e10a76f8af3940b5a18ff24b419f28c'
      testImageTag: 3.4.0-amd64
      testRepository: cp.icr.io/cp/noi-int/netcool-integration-util
    netcool:
      backupHost: ''
      backupPort: 0
      backupServer: ''
      connectionMode: SSLAndAuth
      primaryHost: '10.17.15.114'
      primaryPort: 32732
      primaryServer: 'AGG_P'
      secretName: 'noi254secret'
    probe:
      heartbeatInterval: 10
      rulesConfigmap: ''
      sslServerCommonName: 'noi254-proxy.pink.svc'
      kafka:
        client:
          consumer:
            groupId: 'test-consumer-group'
          saslPlainMechanism: true
          securityProtocol: 'SASL_SSL'
          ssl:
            keyStoreSecretName: ''
            trustStoreSecretName: ''
        connection:
          brokers: 'broker-5-77s66rz443glj9c8.kafka.svc04.us-
south.eventstreams.cloud.ibm.com:9093,broker-3-77s66rz443glj9c8.kafka.svc04.us-
south.eventstreams.cloud.ibm.com:9093,broker-1-77s66rz443glj9c8.kafka.svc04.us-
south.eventstreams.cloud.ibm.com:9093,broker-0-77s66rz443glj9c8.kafka.svc04.us-
south.eventstreams.cloud.ibm.com:9093,broker-4-77s66rz443glj9c8.kafka.svc04.us-
south.eventstreams.cloud.ibm.com:9093,broker-2-77s66rz443glj9c8.kafka.svc04.us-
south.eventstreams.cloud.ibm.com:9093'
          topics: 'kafka-java-console-sample-topic'
          zookeeperClient:
            brokerWatch: false
            target: ''
            topicWatch: false
      jsonParserConfig:
        jsonNestedHeader: ''
        jsonNestedPayload: ''
        messageDepth: 3
        messageHeader: ''
        messagePayload: json
      replicaCount: 1
      autoscaling:
        cpuUtil: 60
        enabled: true
        maxReplicas: 5
        minReplicas: 1
      setUIDandGID: true
      locale: en_US.utf8
      secretName: 'cloudes'
      poddisruptionbudget:
        enabled: false
        minAvailable: 1
      messageLevel: debug
    resources:
      limits:
        cpu: 500m
        memory: 512Mi
      requests:
        cpu: 250m
        memory: 256Mi
  license:
    accept: true
  version: 3.0.0
```

Note: Do not replace this yaml file with your configuration. Just take the values for the properties needed.

4. Install the probe. Check `oc get pods` from the command line to see if all pods are up and running.
5. To uninstall the probe:
   a. Go to **Operators > Installed Operator > All Instances**.
   b. Choose the instance to uninstall.

Installing the Kafka instance using cloudctl case commands

1. Unzip case bundle downloaded. These files can be seen:

```
    ibm-netcool-integrations-1.2.0-images.csv
    ibm-netcool-integrations-1.2.0.tgz
    ibm-noi-int-v1.2.0.tar
```

   `ibm-netcool-integrations-1.2.0.tgz` will be used in the `--case` parameter during installation and uninstallation of probes.
2. Unzip `ibm-netcool-integrations-1.2.0.tgz`, unzipped files here will be used at the `--args` parameter during installation and uninstallation of probes.
3. Install the probe using the **`cloudctl`** command:

```
cloudctl case launch  \
  --case case/ibm-netcool-integrations-*.tgz    \
  --namespace $NAMESPACE     \
  --instance <instance name> \
  --inventory netcoolIntegrationsOperator   \
  --action apply-custom-resources  \
  --args "--inputDir $PWD/case --crFile $PWD/case/ibm-netcool-integrations/inventory/
netcoolIntegrationsOperator/files/deploy/crs/$CRFILE.yaml" \
  --tolerance 1
```

4. Check `oc get pods` from the command line to see if all pods are up and running.
5. To uninstall the probe, use the following command:

```
cloudctl case launch  \
  --case case/ibm-netcool-integrations-*.tgz    \
  --namespace $NAMESPACE     \
  --instance <instance name> \
  --inventory netcoolIntegrationsOperator   \
  --action delete-custom-resources  \
  --args "--inputDir $PWD/case --crFile $PWD/case/ibm-netcool-integrations/inventory/
netcoolIntegrationsOperator/files/deploy/crs/$CRFILE.yaml" \
  --tolerance 1
```

Testing with IBM Event Streams for IBM Cloud

1. Start the producer using the following command (your command might vary as this is just an example). Make sure you have changed the sample event.

```
java -jar ./build/libs/kafka-java-console-
sample-2.0.jar "broker-5-77s66rz443glj9c8.kafka.svc04.us-
south.eventstreams.cloud.ibm.com:9093,broker-3-77s66rz443glj9c8.kafka.svc04.us-
south.eventstreams.cloud.ibm.com:9093,broker-1-77s66rz443glj9c8.kafka.svc04.us-
south.eventstreams.cloud.ibm.com:9093,broker-0-77s66rz443glj9c8.kafka.svc04.us-
south.eventstreams.cloud.ibm.com:9093,broker-4-77s66rz443glj9c8.kafka.svc04.us-
south.eventstreams.cloud.ibm.com:9093,broker-2-77s66rz443glj9c8.kafka.svc04.us-
south.eventstreams.cloud.ibm.com:9093" qPh7UVCSAWJrgr5wmiKjj-knQ_tx2fEiS0b0Z4wK7sVp -producer
```

2. Your probe instance should now be up and running.
   a. Check `oc logs po/podname -f` to see the logs.
   b. Verify that your probe has started receiving events from IBM Event Streams for IBM Cloud.
   c. Verify that events sent to NOI Event Lists properly.

3. Check for any errors in the probe logs.

Customizing probe rules in ConfigMap

This section shows how to modify the default rules file in the probe `ConfigMap` using the command line.

Note: The `ConfigMap` will be deleted when the probe instance is deleted. You should keep a copy of any custom rules file in case you need to use them in the future.

- "Getting the ConfigMap name and rules filename" on page 99
- "Extracting and Customizing the rules file" on page 100
- "Patching the Configmap" on page 100
- "Restarting the Probe Pods" on page 100

Getting the ConfigMap name and rules filename

You need to determine the probe instance name and get the `ConfigMap` name that contains the probe rules files. The `ConfigMap` with the `-rules` suffix should contain the rules files.

The following command uses `example-kafkaprobe` as the probe instance name to query the `ConfigMap` using the `app.kubernetes.io/instance` label. The `--namespace` option can also be specified if required:

```
$ kubectl get configmap -l app.kubernetes.io/instance=example-kafkaprobe
NAME                                     DATA   AGE
example-kafkaprobe-probe-mb-kfk-config   8      12d
example-kafkaprobe-probe-mb-kfk-rules    5      12d
```

Note: The rules file(s) name from the `Data` field. You need to extract the files that you want to customize. This example customizes the `message_bus.rules`.

```
$ kubectl describe configmap example-kafkaprobe-probe-mb-kfk-rules
Name:         example-kafkaprobe-probe-mb-kfk-rules
Namespace:    default
Labels:       app.kubernetes.io/component=mb
              app.kubernetes.io/instance=example-kafkaprobe
              app.kubernetes.io/managed-by=Tiller
              app.kubernetes.io/name=probe-mb-kfk
              hdm.ibm.com/chartversion=3.0.0
              hdm.ibm.com/lastreconciled=
              hdm.ibm.com/resourceowner=
              helm.sh/chart=ibm-netcool-probe-messagebus-kafka-prod
              release=example-kafkaprobe
Annotations:  hdm.ibm.com/lastknownstate: 67c4b74cd6d3b5c44718b38e3cd4517b

Data
====
message_bus.rules:
----
<message_bus.rules file content omitted>

message_bus_cbe.rules:
----
<message_bus_cbe.rules file content omitted>

message_bus_netcool.rules:
----
<message_bus_netcool.rules content omitted>

message_bus_wbe.rules:
----
<message_bus_wbe.rules content omitted>

message_bus_wef.rules:
----
<message_bus_wef.rules content omitted>

Events:  <none>
```

Extracting and Customizing the rules file

To customize the `message_bus.rules` file, first extract the file into you local host:

```
kubectl get configmap example-kafkaprobe-probe-mb-kfk-rules -o
jsonpath='{.data.message_bus\.rules}' > message_bus.rules
```

Then edit the file to make any customization. Ensure that the rules file's syntax is correct.

Patching the Configmap

To patch the `ConfigMap`, first create a YAML file of the "new" `ConfigMap` using the command below. Note that the name of the `ConfigMap` must be the same as the original `ConfigMap`. The output of this command is saved into the file: `example-kafkaprobe-probe-mb-kfk-rules-custom.yaml`.

```
kubectl create configmap --dry-run -o yaml \
--from-file=message_bus.rules \
example-kafkaprobe-probe-mb-kfk-rules.yaml > example-kafkaprobe-probe-mb-kfk-rules-custom.yaml
```

Use the `kubectl patch` command to patch the existing `ConfigMap` with the new YAML file (`example-kafkaprobe-probe-mb-kfk-rules-custom.yaml`)

```
kubectl patch configmap example-kafkaprobe-probe-mb-kfk-rules \
--patch "$(cat example-kafkaprobe-probe-mb-kfk-rules-custom.yaml)"
```

Verify that the `ConfigMap` is correctly patched.

```
kubectl describe configmap example-kafkaprobe-probe-mb-kfk-rules-custom
```

Restarting the Probe Pods

Kubernetes pods do not detect any changes to the `ConfigMap` and the probe pods must be restarted to reload the updated `ConfigMap`.

To restart the pod, scale down and scale up the deployment using the `kubectl scale deployment` command as shown below:

```
kubectl scale deployment example-kafkaprobe-probe-mb-kfk --replicas=0
kubectl scale deployment example-kafkaprobe-probe-mb-kfk --replicas=1
```

Verify that the container is restarting and running:

```
$ kubectl get pods -l app.kubernetes.io/instance=example-kafkaprobe
NAME                                              READY   STATUS                  RESTARTS   AGE
example-kafkaprobe-probe-mb-kfk-8684b6b947-jk76q   1/1     Running                 0          50s
```

Deploying the Gateway for Kafka

IBM Netcool/OMNIbus Gateway for Kafka creates a Kafka producer that processes events and alerts from IBM Netcool/OMNIbus ObjectServer and forwards them to Kafka.

Note: The IBM Netcool Operations Insight Event Integrations Operator must be deployed before deploying this integration. See "Installing the IBM Netcool Operations Insight Event Integrations Operator" on page 21.

Installing integration resource using CLI

This method installs the gateway using the CASE launch scripts. The launch script is executed using the `cloudctl` tool and performs the necessary prerequisite checks for each inventory action that is performed.

Note: The launch scripts have been provided with the CASE since version 1.1.0.

Prerequisites

- The IBM Netcool Operations Insight Event Integration Operator is installed in the target namespace.
- The CASE archive is downloaded and unpacked into your local file system.

Process

1. Extract the custom resource template YAML files from the CASE archive.

   ```
   tar -xvf $CASEPATH ibm-netcool-integrations/inventory/netcoolIntegrationsOperatorSetup/
   files/op-cli/deploy/crs
   ```

2. To deploy a new instance of the Gateway for Kafka, review the custom resource YAML template supplied with the operator package, update the parameters if required, and save the configurable parameters in a file, for example `gateways_v1_kafkagateway.yaml`.
3. Run the `apply-custom-resources` action in the launch script and set the updated custom resource YAML file path using the `--crFile` for each custom resource to install. More than one instance of the gateway can be deployed in the same namespace.

   ```
       cloudctl case launch --case $CASEPATH \
        --namespace $NAMESPACE     \
        --inventory netcoolIntegrationsOperator   \
        --action apply-custom-resources  \
        --args "--crFile $CRFILE" \
        --tolerance 1
   ```

   Note: The CR instance name must be unique to avoid conflicts.

Enable Kafka Gateway to authenticate with SASL enabled Kafka Broker

To authenticate with an SASL enabled Kafka broker, you must pre-create a secret to store the required JAAS configuration before installing the gateway. The example below creates a kubernetes secret from file named `kafka-client-jaas.conf` which contains the JAAS configuration in your pre-created namespace. See "Known limitations of the Gateway for Kafka" on page 107.

```
oc create secret generic kafka-client-jaas --namespace <NAMESPACE> \
  --from-file=kafka-client-jaas=<path to kafka-client-jaas.conf>
```

The following example shows an example of JAAS configuration in `kafka-client-jaas.conf`.

```
KafkaClient {
    org.apache.kafka.common.security.plain.PlainLoginModule required
    serviceName="kafka"
    username="<username to authenticate with Kafka broker>"
    password="<password to authenticate with Kafka broker>";
};
```

Installing the Gateway for Kafka

To deploy a new instance of the Gateway for Kafka, see to the table below for the descriptions of each configurable parameter.

Then, update `deploy/crs/gateways_v1_kafkagateway.yaml` accordingly and save the YAML file.

Parameter name	Description
spec.license.accept	The license state of the image being deployed. Enter true to install and use the image. Default value is `false`.

Parameter name	Description
spec.global.image.secretName	Name of the Secret containing the Docker Config to pull the image from a private repository. Leave blank if the probe image already exists in the local image repository or if the Service Account has already been assigned with an Image Pull Secret. The default is `nil`.
spec.global.serviceAccountName	Name of the service account to be used by the probe instance. If the Cluster Administrator has already created a service account in the namespace, specify the name of the service account here. If left blank, the operator will automatically create a new service account in the namespace when it is deployed. This new service account will be removed from the namespace when the probe instance is removed. The default is `nil`.
spec.global.enableNetworkPolicy	Enables NetworkPolicy to allow traffic from and to the pod. The default is `true`.
spec.global.persistence.useDynamicProvisioning	Use Storage Class to dynamically create Persistent Volume and Persistent Volume Claim. The default value is `true`.
spec.global.persistence.storageClassName	Storage Class for dynamic provisioning. The default is `""`.
spec.global.persistence.selector.label	The Persistent Volume Claim Selector label key to refine the binding process when dynamic provisioning is not used. Default value is `nil`.
spec.global.persistence.selector.value	The Persistent Volume Claim Selector label value related to the Persistent Volume Claim Selector label key. Default value is `nil`.
spec.global.persistence.storageSize	Storage size to store Kafka Gateway Store and Forward Files. The default value is `3Gi`.
spec.image.useDefaultOperandImages	Use default operand images. If true, the operator will use default images for the operands ignoring the image settings in the custom resource YAML file. The operator will perform an upgrade with the latest images from the operator. If false, the operator will use the images specified in the custom resource YAML file to deploy the operand. This should be set to false if you are pulling from a private repository and no image mirroring is configured in the cluster. The default is `true`.

Parameter name	Description
spec.image.gateway	Gateway image repository. The image name must be `netcool-gateway-messagebus`. The default is `cp.icr.io/cp/noi-int/netcool-gateway-messagebus@sha256:b0e4d83315ad2dd46c50fbcfa63ec75a6111c643181e9ad12986b00f23bcc237`.
spec.image.utility	The Utility image repository. The image name must be `netcool-integration-util`. The default is `cp.icr.io/cp/noi-int/netcool-integration-util@sha256:97beb12f8660cdc2392507e9e0d42c026b7ff41f3a5c811785621af8a3906e0a`.
spec.image.pullPolicy	Image pull policy. The default is `IfNotPresent`.
spec.netcool.connectionMode	The connection mode to use when connecting to the Netcool/OMNIbus ObjectServer. For details on available connection modes for your environment, see "Known limitations of the Gateway for Kafka" on page 107. Default value is `AuthOnly`.
spec.netcool.primaryServer	The primary Netcool/OMNIbus server the gateway should connect to (required). Usually set to `NCOMS` or `AGG_P`. The default value is `AGG_P`.
spec.netcool.primaryHost	The host of the primary Netcool/OMNIbus Objsect Server hostname (required). Specify the ObjectServer Hostname. The default value is `nil`.
spec.netcool.primaryIP	The primary Netcool/OMNIbus ObjectServer IP address. If specified along with `primaryHost`, a host alias entry will be added. The default value is `nil`.
spec.netcool.primaryPort	The port number of the primary Netcool/OMNIbus ObjectServer (required). The default is `4100`.
spec.netcool.primaryIDUCHost	The primary Netcool/OMNIbus ObjectServer IDUC Host or Service name. Should be set if the primary IDUC host is different from the primary ObjectServer hostname. The default value is `nil`.
spec.netcool.backupServer	The backup Netcool/OMNIbus ObjectServer to connect to. If the `backupServer`, `backupHost` and `backupPort` parameters are defined in addition to the `primaryServer`, `primaryHost`, and `primaryPort` parameters, the gateway will be configured to connect to a virtual ObjectServer pair called `AGG_V`. The default value is `nil`.

Parameter name	Description
spec.netcool.backupHost	The host of the backup Netcool/OMNIbus ObjectServer. Specify the backup ObjectServer hostname. The default value is `nil`.
spec.netcool.backupIP	The backup Netcool/OMNIbus ObjectServer IP address. If specified along with primaryHost, a host alias entry will be added. The default value is `nil`.
spec.netcool.backupPort	The port of the backup Netcool/OMNIbus ObjectServer. The default value is `4100`.
spec.netcool.backupIDUCHost	The backup Netcool/OMNIbus ObjectServer IDUC host or service name. Should be set if the backup IDUC host is different from the backup ObjectServer hostname. The default value is `nil`.
spec.netcool.secretName	A pre-created secret for `AuthOnly`, `SSLOnly` or `SSLAndAuth` connection mode. Certain fields are required depending on the connection mode. The default value is `nil`.
spec.netcool.waitTimeout	The duration (in minutes) to wait for the ObjectServer to become available. The default is `0` (disabled).
spec.gateway.messageLevel	Gateway log message level. The default is `warn`.
spec.gateway.enableTransportDebugLog	Set to true to enable common Transport Module debug logging for troubleshooting purposes. The default value is `false`.
spec.gateway.setUIDandGID	When set to true, the operator will specify the UID and GID values for the netcool user else the netcool user will not be assigned any UID or GID by the operator. When deploying in an environment that is enforced by SecurityContextConstraints (SCC) that requires the process to have a fixed UID and GID value, this parameter must be set to true. Otherwise, if deploying in an environment enforced by an SCC that requires the process to support arbitrary UIDs, for example in the Openshift restricted SCC, this parameter must be set to false. The default value is `false`.
spec.gateway.locale	Environment locale setting. Used as the LC_ALL environment variable. The default is `en_US.utf8`.
spec.gateway.secretName	Name of secret which contains authentication credentials for Kafka Transport. This secret must contain `kafka_username` and `kafka_password` keys. The default is `nil`.

Parameter name	Description
spec.gateway.reader.logOSSql	Log all SQL sent to ObjectServer in debug mode. The default is `false`.
spec.gateway.reader.statusTableName	Specifies the target table for `alerts.status`. This field is mandatory. The default is `alerts.status`.
spec.gateway.reader.journalTableName	Specifies the target table for `alerts.journal`. This field is mandatory. The default is `alerts.journal`.
spec.gateway.reader.detailsTableName	Specifies the target table for `alerts.details`. This field is mandatory. The default is `alerts.details`.
spec.gateway.kafka.connection.zookeeperClient.target	Use this property to specify the ZooKeeper endpoint for example, `zookeeper:2181` The default is `nil`.
spec.gateway.kafka.connection.zookeeperClient.topicWatch	Use this property to enable the ZooKeeper topic watch service. The default is `true`.
spec.gateway.kafka.connection.zookeeperClient.brokerWatch	Use this property to enable the ZooKeeper broker watch service. The default is `true`.
spec.gateway.kafka.connection.brokers	Use this property to specify the broker endpoints in a comma-separated list for example, `PLAINTEXT://kafkaserver1:9092,PLAINTEXT://kafkaserver2:9092`. The default is `nil`.
spec.gateway.kafka.connection.topics	Use this property to specify topics in a comma-separated list for example, `topicABC,topicXYZ`. The default is `nil`.
spec.gateway.kafka.client.securityProtocol	The security protocol to be used. The default is `nil`.
spec.gateway.kafka.client.ssl.trustStoreSecretName	Secret name for the TrustStore file and its password. The default is `nil`.
spec.gateway.kafka.client.sasl.jaasConfigSecretName	Secret name which contains the Kafka producer JAAS configuration file. See "Enable Kafka Gateway to authenticate with SASL enabled Kafka Broker" on page 101. The default is `nil`.

Parameter name	Description

spec.gateway.kafka.client.producer.configs	A multi-line string parameter to configure additional Kafka producer configurations. Each entry must not contain space. For more details when installing or configuring this parameter on the OLM UI see "Known limitations of the Gateway for Kafka" on page 107.
	The default is
	`ssl.protocol=TLSv1.2`
	`ssl.enabled.protocols=TLSv1.2`
	`sasl.mechanism=PLAIN`
	`client.id=KafkaGateway`
	`acks=all`
	`max.block.ms=60000`
	`retries=0`
	`batch.size=16384`
	`buffer.memory=33554432`
	`key.serializer=org.apache.kafka.common.serial`
	`ization.StringSerializer`
	`value.serializer=org.apache.kafka.common.seri`
	`alization.StringSerializer`

spec.gateway.kafka.liveness.criterion	Use this property to control the periodic Kafka liveness check feature. This property takes the following values: `TOPIC`: Enables liveness check. `NONE`: Disables liveness check. The default value is `TOPIC`.
spec.gateway.kafka.liveness.checkInterval	Use this property to configure the interval (in seconds) to run the Kafka liveness check. The range of valid values is 10 to 1000 seconds The default value is `20`.
spec.gateway.kafka.liveness.checkTimeout	Use this property to configure the time (in seconds) the transport allows for the liveness check operation to complete. If the operation does not complete within this period, the transport regards it as a negative Kafka liveness status. The range of valid values is 5 to 60 seconds. The default value is 5.
spec.gateway.kafka.liveness.lockSendUntilReconnection	Use this property to configure the transport to lock negative liveness status and stop message processing. This property takes the following values: TRUE: Locks the negative liveness status after a failed liveness check report. FALSE: Does not lock the negative liveness status. This used for debugging purposes only. The default value is TRUE.
spec.resources.limits.cpu	CPU resource limits. The default is `500m`.

Parameter name	Description
spec.resources.limits.memory	Memory resource limits. The default is 512Mi.
spec.resources.limits.ephemeral-storage	Ephemeral storage limits. The default is 500Mi.
spec.resources.requests.cpu	CPU resource requests. The default is 250m.
spec.resources.requests.memory	Memory resource requests. The default is 256Mi.
spec.resources.requests.ephemeral-storage	Ephemeral storage requests. The default is 100Mi.
spec.arch	Worker node architecture. Fixed to amd64.

Then, deploy the custom resource using the updated YAML file:

```
kubectl apply -f deploy/crs/gateways_v1_kafkagateway.yaml --namespace <NAMESPACE>
```

Known limitations of the Gateway for Kafka

This section describes various limitations that currently exist.

The Gateway for Kafka has the following known limitations:

- If you opt to use the OLM UI to install or configure the Gateway for Kafka, you must do so using YAML view instead of the Form view. There is a known limitation in the Form view whereby it changes the Kafka Client Producer Configuration from a multi-line string into a single-line string which is not expected by the gateway and the text field does not accept new line characters.
- Currently the Gateway for Kafka only supports `SASL/PLAIN` which is a simple username/password authentication mechanism. The gateway does not support `SASL/GSSAPI` which uses Kerberos for authentication.

Deploying the Probe for Pulsar Integration

IBM Netcool/OMNIbus Probe for Pulsar Integrations creates a Pulsar client, subscribes and processes events from a Pulsar broker.

Note: The IBM Netcool Operations Insight Event Integrations Operator must be deployed before deploying this integration. See "Installing the IBM Netcool Operations Insight Event Integrations Operator" on page 21.

To deploy a new instance of the probe, review the custom resource YAML template supplied with the operator package, update the parameters if required, and save the configurable parameters in a file, for example `probes_v1_pulsarprobe.yaml`.

Then deploy the custom resource using the updated file:

```
kubectl apply -f probes_v1_pulsarprobe.yaml --namespace <NAMESPACE>
```

Note: If you are installing using the CLI, you can get the custom resource Yaml file from the `ibm-netcool-integrations/inventory/netcoolIntegrationsOperatorSetup/files/op-cli/deploy/crs` directory in our CASE archive. If you are installing using the Operator Lifecycle Manager (OLM) Web console, the custom resource is provided in the YAML view.

Pre-installation steps

Before installing the probe, you must perform the following tasks:

1. "Securing communications with the Pulsar Broker" on page 108
2. "Securing communications with the ObjectServer" on page 108

3. “(Optional) Creating a persistent volume claim to provide probe configuration” on page 108

Securing communications with the Pulsar Broker

To secure Probe and Pulsar Broker communication, you must pre-create a secret to store the required certificates before installing the probe. The example below creates a kubernetes secret contains the trusted TLS certificate, client certificates and authentication plugin class name in your pre-created namespace.

```
kubernetes create secret generic message-bus-pulsar-cert --namespace <NAMESPACE> \
  --from-file=tlsTrustCertFile=certs/ca.cert.pem \
  --from-file=tlsCertFile=certs/admin.cert.pem \
  --from-file=tlsKeyFile=certs/admin.key-pk8.pem \
  --from-literal=authPluginClassName=org.apache.pulsar.client.impl.auth.AuthenticationTls
```

Securing communications with the ObjectServer

See Secure probe and ObjectServer communication requirements.

(Optional) Creating a persistent volume claim to provide probe configuration

To allow the probe to access the custom probe files in Persistent Volume (PV), you must pre-create a Persistent Volume Claim (PVC) . The PVC must have access mode of `ReadWriteMany` (RWX). You must determine the amount of storage available to the PVC according to your total size of files that you want to store in PV.

Configurable parameters

The following table describes the configurable parameters for this probe and lists their default values.

Parameter name	Description
spec.license.accept	The license state of the image being deployed. Enter `accept` to install and use the image. The default is `not accepted`.
spec.version	Probe application version. Accepted versions are `1.1.0`, `1.2.0` or `1.3.0`. The default is `1.3.0`.
spec.helmValues.global.image.secretName	Name of the Secret containing the Docker Config to pull the image from a private repository. Leave blank if the probe image already exists in the local image repository or if the Service Account has already been assigned with an Image Pull Secret. The default is `nil`.
spec.helmValues.global.serviceAccountName	Name of the service account to be used by the probe instance. If the Cluster Administrator has already created a service account in the namespace, specify the name of the service account here. If left blank, the operator will automatically create a new service account in the namespace when it is deployed. This new service account will be removed from the namespace when the instance is removed. The default is `nil`.
spec.helmValues.global.enableNetworkPolicy	Enables NetworkPolicy to allow traffic from and to the pod. The default is `true`.

Parameter name	Description
spec.helmValues.image.useDefaultOperandImages	Use default operand images. If `true`, the operator will use default images for the operands ignoring the image settings in the custom resource YAML file. The operator will perform an upgrade with the latest images from the operator. If `false`, the operator will use the images specified in the custom resource YAML file to deploy the operand. This should be set to `false` if you are pulling from a private repository and no image mirroring is configured in the cluster. Default is `true`.
spec.helmValues.image.probe	Probe image repository. The image name must be `netcool-probe-messagebus`. The default is `cp.icr.io/cp/noi-int/netcool-probe-messagebus@sha256:7d524c2bebd30640bbbc2ed89367800b65b6cfdc2c53a1e5785577284af36956`.
spec.helmValues.image.utility	The Utility image repository. The image name must be `netcool-integration-util`. The default is `cp.icr.io/cp/noi-int/netcool-integration-util@sha256:97beb12f8660cdc2392507e9e0d42c026b7ff41f3a5c811785621af8a3906e0a`.
spec.helmValues.image.pullPolicy	Image pull policy. The default is `IfNotPresent`.
spec.helmValues.netcool.connectionMode	The connection mode to use when connecting to the Netcool/OMNIbus ObjectServer. Refer to Securing Probe and ObjectServer Communications section for more details. **Note**: Refer to limitations section for more details on available connection modes for your environment. The default is `default`.
spec.helmValues.netcool.primaryServer	The primary Netcool/OMNIbus server the probe should connect to (required). Usually set to NCOMS or AGG_P. The default is nil.
spec.helmValues.netcool.primaryHost	The host of the primary Netcool/OMNIbus ObjectServer (required). Specify the ObjectServer Hostname or IP address. The default is nil.
spec.helmValues.netcool.primaryPort	The port number of the primary Netcool/OMNIbus server (required). The default is nil.
spec.helmValues.netcool.backupServer	The backup Netcool/OMNIbus ObjectServer to connect to. If the backupServer, backupHost and backupPort parameters are defined in addition to the primaryServer, primaryHost, and primaryPort parameters, the probe will be configured to connect to a virtual ObjectServer pair called `AGG_V`. The default is `nil`.

Parameter name	Description
spec.helmValues.netcool.backupHost	The host of the backup Netcool/OMNIbus ObjectServer. Specify the ObjectServer Hostname or IP address. The default is `nil`.
spec.helmValues.netcool.backupPort	The port of the backup Netcool/OMNIbus ObjectServer. The default is `4100`.
spec.helmValues.probe.messageLevel	Probe log message level. The default is `warn`.
spec.helmValues.probe.setUIDandGID	When set to `true`, the operator will specify the UID and GID values for the netcool user else the netcool user will not be assigned any UID or GID by the operator. When deploying in an environment that is enforced by SecurityContextConstraints (SCC) that requires the process to have a fixed UID and GID value, this parameter must be set to `true`. Otherwise, if deploying in an environment enforced by an SCC that requires the process to support arbitrary UIDs, for example in the Openshift `restricted` SCC, this parameter must be set to `false`. Default value is `false`.
spec.helmValues.probe.heartbeatInterval	Specifies the frequency (in seconds) with which the probe checks the status of the host server. The default is `10`.
spec.helmValues.probe.inactivity	Time (in seconds) that the probe allows the port to receive no incoming data before disconnecting. The default is `0` (disabled).
spec.helmValues.probe.maxBufferedReaderLength	Maximum probe buffered reader line length in bytes. The default is `16384`.
spec.helmValues.probe.enableTransportDebugLog	Set to `true` to enable common Transport Module debug logging for troubleshooting purposes. Default value is `false`.
spec.helmValues.probe.configPVC.name	(Optional) Specifies a pre-created Persistent Volume Claim name to access custom probe files in Persistent Volume. If left unset, the probe will use the default Message Bus probe files. The default is `nil`
spec.helmValues.probe.configPVC.rulesFile	(Optional) Specifies the path to the main probe rules file in the persistent volume. If left unset, the probe will use the default Message Bus probe rules files. This field will override the selection made for the `probe.rulesConfigmap` parameter. The default is `nil`

Parameter name	Description
spec.helmValues.probe.configPVC.parserConfigFile	(Optional) Specifies the path to the JSON parser configuration file in the persistent volume. If left unset, the probe will use the default JSON parser configuration. This field will override the selection made for the `spec.helmValues.probe.jsonParserConfig.messagePayload`, `spec.helmValues.probe.jsonParserConfig.messageHeader`, `spec.helmValues.probe.jsonParserConfig.jsonNestedPayload`, `spec.helmValues.probe.jsonParserConfig.jsonNestedHeader` and `spec.helmValues.probe.jsonParserConfig.messageDepth` parameters. The default is `nil`.
spec.helmValues.probe.rulesConfigmap	(Optional) Specifies a customised ConfigMap to be used for the rules files. This field will override the selection made for the `probe.rulesFile` parameter. This field must be left blank if not required. The default is `nil`.
spec.helmValues.probe.jsonParserConfig.messagePayload	Specifies the JSON tree to be identified as message payload from the notification (pulsar consumer) channel. See example for more details on how to configure the Probe's JSON parser. The default is `json`.
spec.helmValues.probe.jsonParserConfig.messageHeader	(Optional) Specifies the JSON tree to be identified as message header from the notification (pulsar consumer) channel. Attributes from the headers will be added to the generated event. The default is `nil`.
spec.helmValues.probe.jsonParserConfig.jsonNestedPayload	(Optional) Specifies the JSON tree within a nested JSON or JSON string to be identified as message payload from the notification (pulsar consumer) channel. The `probe.jsonParserConfig.messagePayload` parameter must be set to point to the attribute containing the JSON String. The default is `nil`.
spec.helmValues.probe.jsonParserConfig.jsonNestedHeader	(Optional) Specifies the JSON tree within a nested JSON or JSON string to be identified as message header from the notification (pulsar consumer) channel. The `probe.jsonParserConfig.messageHeader` parameter must be set to point to the attribute containing the JSON String. The default is `nil`.
spec.helmValues.probe.jsonParserConfig.messageDepth	Specifies the number of JSON child node levels in the message to traverse during parsing. The default is 3.

Parameter name	Description
spec.helmValues.probe.selfMonitoring.discardHeartbeatEvent	Set this value to `true` if you want to discard the ProbeWatch Heartbeat event. Set this value to `false` if you want to forward the ProbeWatch Heartbeat event to the ObjectServer. The default is `false`.
spec.helmValues.probe.selfMonitoring.populateMasterProbeStat	Set this value to `true` to enable a new probe metrics event to be generated by using the genevent function, which is defined within the case "Heartbeat ..." statement. The event data consists a set of OplStats probe metrics, which are forwarded to the `master.probestats` table that was created when you ran the `probestats.sql` script. Set this value to `false` if you do not want to generate this event for insertion into the `master.probestats` table. The default is `false`.
spec.helmValues.probe.selfMonitoring.logProbeStat	Set this value to `true` if you want to record the OplStats metrics in the probe log file. Details are logged at the INFO level. The metric details that are logged are defined in the case "Heartbeat ..." statement. Set this value to `false` if you do not want to record the metrics in the log file. The default is `false`.
spec.helmValues.probe.selfMonitoring.generateThresholdEvents	Set this value to `true` if you want to generate threshold events that indicate when a particular probe metric violates a defined threshold. By default, no code is provided for threshold events within this rules file because individual preferences can vary widely. If you require threshold events, you must first decide which thresholds you want to monitor. Then, within the case "Heartbeat ..." statement, provide the code for generating threshold events. The default is `false`.
spec.helmValues.probe.configs	A multiline string parameter to configure additional probe properties. Each entry must not contain space. The default is `ProbeWatchHeartbeatInterval:0 EventLoadProfiling:'true'`.
spec.helmValues.probe.pulsar.client.serviceUrl	Service URL provider for Pulsar service. The default is `nil`.
spec.helmValues.probe.pulsar.client.enableTls	Enable TLS connection between the probe with Pulsar broker. The default is `false`.
spec.helmValues.probe.pulsar.client.enableAuth	Enable the probe to authenticate with Pulsar broker. The default is `false`.

Parameter name	Description
spec.helmValues.probe.pulsar.client.secretName	A pre-created secret contains trusted TLS certificate file, name of the authentication plugin or the parameters for authentication plugin. The default is `nil`.
spec.helmValues.probe.pulsar.client.tlsAllowInsecureConnection	To allow the probe accepts untrusted TLS certificate from Pulsar broker. The default is `false`.
spec.helmValues.probe.pulsar.client.tlsHostnameVerificationEnable	To enable TLS hostname verification. The default is `false`.
spec.helmValues.probe.pulsar.consumer.topicNames	Topic name. The default is `nil`.
spec.helmValues.probe.pulsar.consumer.subscriptionName	Subscription name. The default is `nil`.
spec.helmValues.probe.pulsar.consumer.subscriptionType	The Pulsar consumer subscription type. The default is `Exclusive`.
spec.helmValues.probe.replicaCount	Number of deployment replicas. Omitted when `autoscaling.enabled` set to `true`. The default is `1`.
spec.helmValues.probe.autoscaling.enabled	Set to `false` to disable auto-scaling. The default is `true`.
spec.helmValues.probe.autoscaling.minReplicas	Minimum number of probe replicas. The default is `1`.
spec.helmValues.probe.autoscaling.maxReplicas	Maximum number of probe replicas. The default is `5`.
spec.helmValues.probe.autoscaling.cpuUtil	The target CPU utilization (in percentage). Example: `60` for 60% target utilization. The default is `60`.
spec.helmValues.probe.poddisruptionbudget.enabled	Set to `true` to enable Pod Disruption Budget to maintain high availability during a node maintenance. Administrator role or higher is required to enable Pod Disruption Budget on clusters with Role Based Access Control. The default is `false`.
spec.helmValues.probe.poddisruptionbudget.minAvailable	The number of minimum available pods during node drain. Can be set to a number or percentage, eg: 1 or 10%. Caution: Setting to 100% or equal to the number of replicas may block node drains entirely. The default is `1`.
spec.helmValues.resources.limits.cpu	CPU resource limits. The default is `500m`.
spec.helmValues.resources.limits.memory	Memory resource limits. The default is `512Mi`.

Parameter name	Description
spec.helmValues.resources.limits.ephemeral-storage	Ephemeral storage limits. The default is `500Mi`.
spec.helmValues.resources.requests.cpu	CPU resource requests. The default is `250m`.
spec.helmValues.resources.requests.memory	Memory resource requests. The default is `256Mi`.
spec.helmValues.resources.requests.ephemeral-storage	Ephemeral storage requests. The default is `100Mi`.
spec.helmValues.arch	Worker node architecture. Fixed to `amd64`.

Post-installation steps

If you want to store rules and/or JSON parser configuration files in the PV and `spec.helmValues.probe.configPVC.name` is set, you must perform the following steps:

1. "Providing configuration files during probe pod initialization" on page 114
2. "Updating configuration files in the persistent volume" on page 114

Providing configuration files during probe pod initialization

The command below shows how to add custom probe files into the PV from local directory during probe pod initialization. You only perform this step once for a new PVC. The probe will not start until all files are successfully transferred.

```
oc rsync  ./ <probe pod>:/home/netcool/etc/custom_probe_files -c custom-probefiles-wait --
strategy=tar
```

Updating configuration files in the persistent volume

The command below shows how to update custom probe files stored in PV from local directory when probe pod is running. The local directory contains the updated probe files.

```
oc rsync  ./ <probe pod>:/home/netcool/etc/custom_probe_files --strategy=tar
```

You must run the commands below to scale down the number of replicas to zero and scale up to the desired replica count to ensure that the probe pods reload with the updated configuration file and rules files from the PV.

```
oc scale deployment <probe deployment> --replicas=0
```

```
oc scale deployment <probe deployment> --replicas=<desired replica count>
```

Updating probe rules files

If you are only updating the custom rules files in the persistent volume, run the command below to trigger the probe to reload the updated rules files. You must run this command on every pod if there are more than one probe pod replicas so that all pods run with the same rules.

```
oc exec <probe pod>  \
-n <namespace>   -it \
-- curl -X PUT http://<probe pod>:8080/probe/common/reloadrules \
-H 'Content-type:application/json' \
-d  '{"reloadrulesflag":1}'
```

Deploying the Gateway for Pulsar Integration

IBM Netcool/OMNIbus Gateway for Pulsar creates a Pulsar producer that processes events and alerts from IBM Netcool/OMNIbus ObjectServer and forwards them to Pulsar.

Note: The IBM Netcool Operations Insight Event Integrations Operator must be deployed before deploying this integration. See "Installing the IBM Netcool Operations Insight Event Integrations Operator" on page 21.

To deploy a new instance of Pulsar Gateway, refer to the table below for the description of each configurable parameter. Then, update `gateways_v1_pulsargateway.yaml` accordingly and save the YAML file.

Note: If you are installing using the CLI, you can get the custom resource Yaml file from the `ibm-netcool-integrations/inventory/netcoolIntegrationsOperatorSetup/files/op-cli/deploy/crs` directory in our CASE archive. If you are installing using the Operator Lifecycle Manager (OLM) Web console, the custom resource is provided in the YAML view.

Pre-installation steps

Before installing the gateway, you must perform the following tasks:

1. "Securing gateway and pulsar broker communication" on page 115

Securing gateway and pulsar broker communication

To secure Gateway and Pulsar Broker communication, you must pre-create a secret to store the required certificates before installing the gateway. The example below creates a kubernetes secret contains the trusted TLS certificate, client certificates and authentication plugin class name in your pre-created namespace.

```
kubernetes create secret generic gateway-pulsar-cert --namespace <NAMESPACE> \
  --from-file=tlsTrustCertFile=certs/ca.cert.pem \
  --from-file=tlsCertFile=certs/admin.cert.pem \
  --from-file=tlsKeyFile=certs/admin.key-pk8.pem \
  --from-literal=authPluginClassName=org.apache.pulsar.client.impl.auth.AuthenticationTls
```

Configurable parameters

The following table describes the configurable parameters for this probe and lists their default values.

Parameter name	Description
spec.license.accept	The license state of the image being deployed. Enter `true` to install and use the image. Default value is `false`.
spec.version	Gateway application version. Accepted versions are `1.0.0` and `1.1.0`. Default value is `1.1.0`.
spec.helmValues.global.image.secretName	Name of the Secret containing the Docker Config to pull the image from a private repository. Leave blank if the gateway image already exists in the local image repository or if the Service Account has already been assigned with an Image Pull Secret. The default is `nil`.

Parameter name	Description
spec.helmValues.global.serviceAccountName	Name of the service account to be used by the helm chart. If the Cluster Administrator has already created a service account in the namespace, specify the name of the service account here. If left blank, the chart will automatically create a new service account in the namespace when it is deployed. This new service account will be removed from the namespace when the chart is removed. The default is `nil`.
spec.helmValues.global.enableNetworkPolicy	Enables NetworkPolicy to allow traffic from and to the pod. The default is `true`.
spec.helmValues.image.useDefaultOperandImages	Use default operand images. If `true`, the operator will use default images for the operands ignoring the image settings in the custom resource YAML file. The operator will perform an upgrade with the latest images from the operator. If `false`, the operator will use the images specified in the custom resource YAML file to deploy the operand. This should be set to `false` if you are pulling from a private repository and no image mirroring is configured in the cluster. Default is `true`.
spec.helmValues.image.gateway	Gateway image repository. The image name must be `netcool-gateway-messagebus`. The default is `cp.icr.io/cp/noi-int/netcool-gateway-messagebus@sha256:b0e4d83315ad2dd46c50fbcfa63ec75a6111c643181e9ad12986b00f23bcc237`.
spec.helmValues.image.utility	The Utility image repository. The image name must be `netcool-integration-util`. The default is `cp.icr.io/cp/noi-int/netcool-integration-util@sha256:97beb12f8660cdc2392507e9e0d42c026b7ff41f3a5c811785621af8a3906e0a`.
spec.helmValues.image.pullPolicy	Image pull policy. The default is `IfNotPresent`.
spec.helmValues.netcool.connectionMode	The connection mode to use when connecting to the Netcool/OMNIbus ObjectServer. Refer to Securing Probe and ObjectServer Communications section for more details. **Note**: Refer to limitations section for more details on available connection modes for your environment. The default is `AuthOnly`.
spec.helmValues.netcool.primaryServer	The primary Netcool/OMNIbus ObjectServer the gateway should connect to (required). Usually set to NCOMS or AGG_P. The default is `AGG_P`.

Parameter name	Description
spec.helmValues.netcool.primaryHost	The host of the primary Netcool/OMNIbus ObjectServer (required). Specify the ObjectServer hostname. The default is `nil`.
spec.helmValues.netcool.primaryIP	The primary Netcool/OMNIbus ObjectServer IP address. If specified along with primaryHost, a host alias entry will be added. The default is `nil`.
spec.helmValues.netcool.primaryPort	The port number of the primary Netcool/OMNIbus ObjectServer (required). The default is `4100`.
spec.helmValues.netcool.primaryIDUCHost	The primary Netcool/OMNIbus ObjectServer IDUC Host or Service name. Should be set if the primary IDUC host is different from the primary ObjectServer hostname. The default is `nil`.
spec.helmValues.netcool.backupServer	The backup Netcool/OMNIbus ObjectServer to connect to. If the backupServer, backupHost and backupPort parameters are defined in addition to the primaryServer, primaryHost, and primaryPort parameters, the gateway will be configured to connect to a virtual ObjectServer pair called `AGG_V`. The default is `nil`.
spec.helmValues.netcool.backupHost	The host of the backup Netcool/OMNIbus ObjectServer. Specify the backup ObjectServer hostname. The default is `nil`.
spec.helmValues.netcool.backupIP	The backup Netcool/OMNIbus ObjectServer IP address. If specified along with primaryHost, a host alias entry will be added. The default is `nil`.
spec.helmValues.netcool.backupPort	The port of the backup Netcool/OMNIbus ObjectServer. The default is `4100`.
spec.helmValues.netcool.backupIDUCHost	The backup Netcool/OMNIbus ObjectServer IDUC Host or Service name. Should be set if the backup IDUC host is different from the backup ObjectServer hostname. The default is `nil`.
spec.helmValues.netcool.secretName	A pre-created secret for AuthOnly, SSLOnly or SSLAndAuth connection mode. Certain fields are required depending on the connection mode. The default is `nil`.
spec.helmValues.netcool.waitTimeout	The duration (in minutes) to wait for the ObjectServer to become available. The default is `0` (disabled).
spec.helmValues.gateway.messageLevel	Gateway log message level. The default is `warn`.
spec.helmValues.gateway.enableTransportDebugLog	(Optional) Set to true to enable the common Transport Module debug logging. The default is `false`.

Parameter name	Description
spec.helmValues.gateway.setUIDandGID	When set to true, pods are run with specific UID and GID values. Otherwise, when set to false the pods runs with an arbitrary UID, for example when deploying in the OpenShift native restricted SecurityContextConstraints. The default is `false`.
spec.helmValues.gateway.locale	Environment locale setting. Used as the LC_ALL environment variable. The default is `en_US.utf8`.
spec.helmValues.gateway.reader.logOSSql	Log all SQL sent to ObjectServer in debug mode. The default is `false`.
spec.helmValues.gateway.reader.statusTableName	Specifies the target table for alerts.status. This field is mandatory. The default is `alerts.status`.
spec.helmValues.gateway.reader.journalTableName	Specifies the target table for alerts.journal. This field is mandatory. The default is `alerts.journal`.
spec.helmValues.gateway.reader.detailsTableName	Specifies the target table for alerts.details. This field is mandatory. The default is `alerts.details`.
spec.helmValues.gateway.pulsar.client.serviceUrl	Service URL provider for Pulsar service. The default is `nil`.
spec.helmValues.gateway.pulsar.client.enableTls	Enable TLS connection between the gateway with Pulsar broker. The default is `false`.
spec.helmValues.gateway.pulsar.client.enableAuth	Enable the gateway to authenticate with Pulsar broker. The default is `false`.
spec.helmValues.gateway.pulsar.client.secretName	The secret contains trusted TLS certificate file, name of the authentication plugin or the parameters for authentication plugin. The default is `nil`.
spec.helmValues.gateway.pulsar.client.tlsAllowInsecureConnection	To allow the gateway accepts untrusted TLS certificate from Pulsar broker. The default is `false`.
spec.helmValues.gateway.pulsar.client.tlsHostnameVerificationEnable	To enable TLS hostname verification. The default is `false`.
spec.helmValues.gateway.pulsar.producer.topicName	Topic name. The default is `nil`.
spec.helmValues.gateway.pulsar.producer.producerName	Producer name. The default is `nil`.

Parameter name	Description

spec.helmValues.gateway.pulsar.producer.configs	(Optional) A multiline string parameter to configure additional Pulsar Producer Configurations. Each entry must not contain space. The default is:
	`sendTimeoutMs=30000`
	`blockIfQueueFull=false`
	`maxPendingMessages=1000`
	`batchingEnabled=true`
	`batchingMaxPublishDelayMicros=1000`
	`batchingMaxMessages=1000`
	`compressionType=NONE`

spec.helmValues.resources.limits.cpu	CPU resource limits. The default is `500m`.
spec.helmValues.resources.limits.memory	Memory resource limits. The default is `512Mi`.
spec.helmValues.resources.limits.ephemeral-storage	Ephemeral storage limits. The default is `500Mi`
spec.helmValues.resources.requests.cpu	CPU resource requests. The default is `250m`.
spec.helmValues.resources.requests.memory	Memory resource requests. The default is `256Mi`.
spec.helmValues.resources.requests.ephemeral-storage	Ephemeral storage requests. The default is `100Mi`.
spec.helmValues.arch	Worker node architecture. Fixed to `amd64`.

Deploying the Generic Webhook Probe

IBM Netcool/OMNIbus Probe for Generic Webhook creates webhook endpoint to receive notification in the form of HTTP POST requests with JSON payloads from event source.

Note: The IBM Netcool Operations Insight Event Integrations Operator must be deployed before deploying this integration. See "Installing the IBM Netcool Operations Insight Event Integrations Operator" on page 21.

To deploy a new instance of the Generic Webhook Probe, review the custom resource YAML template supplied with the operator package, update the parameters if required, and save the configurable parameters in a file, for example `probes_v1_webhookprobe.yaml`.

Then deploy the custom resource using the updated file:

```
kubectl apply -f deploy/crs/probes_v1_webhookprobe.yaml --namespace <NAMESPACE>
```

Note: If you are installing using the CLI, you can get the custom resource Yaml file from the `ibm-netcool-integrations/inventory/netcoolIntegrationsOperatorSetup/files/op-cli/deploy/crs` directory in our CASE archive. If you are installing using the Operator Lifecycle Manager (OLM) Web console, the custom resource is provided in the YAML view.

Securing the probe to ObjectServer connection

There are several mechanisms to secure Netcool/OMNIbus system. Authentication can be used to restrict user access while Secure Sockets Layer (SSL) protocol can be used for different levels of encryption.

The probe connection mode depends on the server component configuration. Check with your Netcool/OMNIbus Administrator whether the server is configured with either secured mode enabled without SSL, SSL enabled with secured mode disabled, or secured mode enabled with SSL protected communications.

For more details on running the ObjectServer in secured mode, see Running the ObjectServer in secure mode.

For more details on SSL protected communications, see Using SSL for client and server communications.

The probe must be configured according to the server components set up in order to establish a secured connection with or without SSL. This can be configured by setting the `netcool.connectionMode` parameter with one of these options:

- `Default` - This is the default mode. Use this mode to connect with the ObjectServer with neither secure mode nor SSL.
- `AuthOnly` - Use this mode when the ObjectServer is configured to run in secured mode without SSL.
- `SSLOnly` - Use this mode when the ObjectServer is configured with SSL without secure mode.
- `SSLAndAuth` - Use this mode the ObjectServer is configured with SSL and secure mode.

If you are using the `Default` mode, you can skip the these steps and proceed configuring the probe with your ObjectServer connection details.

To secure the communications between probe clients and the ObjectServer, refer to the topics below according to your ObjectServer setup for required pre-installation tasks.

"For on-premise Netcool/OMNIbus ObjectServer" on page 120

"For Netcool Operations Insight (NOI) on RedHat Openshift Container Platform (OCP)" on page 122

Determine files required for the secret

For `AuthOnly`, `SSLOnly` and `SSLAndAuth`, a `Secret` is required to support the secured connection prior installing the probe on OCP. The secret name should be set in the `netcool.secretName` parameter when configuring the probe or left unset if you want to use the default connection mode.

Several fields are required in the Secret depending on the connection mode that will be configured in the operand custom resource.

For secured mode (`AuthOnly`), the following fields are required:

- `AuthUserName`: Specify the username to authenticate with the ObjectServer.
- `AuthPassword`: Specify the password for the specified user.

For SSL enabled mode (`SSLOnly`), the following fields are required:

- `tls.crt`: The ObjectServer TLS certificate to import into the probe truststore.
- `ca.crt`: The ObjectServer Certificate Authority (CA) certificate to import in the probe truststore. If there is a common CA certificate that is used to signed the ObjectServer certificates, then you can opt to just import this CA.

For secured and SSL enabled mode (`SSLAndAuth`), the following fields are required:

- `AuthUserName`: Specify the username to authenticate with the ObjectServer.
- `AuthPassword`: Specify the password of the specified user.
- `tls.crt`: The ObjectServer TLS certificate to import into the probe truststore.
- `ca.crt`: The ObjectServer Certificate Authority (CA) certificate to import into the probe truststore. If there is a common CA certificate that is used to sign the ObjectServer certificates, then you can opt to just import this CA.

For on-premise Netcool/OMNIbus ObjectServer

Preparing credentials for authentication

This section is only applicable if you are using either the `AuthOnly` or `SSLAndAuth` connection mode.

For the probe to connect with a secured ObjectServer, the probe must be configured with a valid credential otherwise the ObjectServer will reject the connection. The credentials will be set in a `Secret` for the pods to retrieve the secret and configure the probe. Follow the steps below to prepare the required files prior creating the `Secret`.

1. Contact your Netcool/OMNIbus Administrator to obtain the credentials that should be used by probe clients to authenticate with the ObjectServer. Optionally, you can also request for an encrypted password with its encryption key file from the administrator to be used in the `Secret`.
2. Create a temporary workspace directory if necessary. The following steps use `/tmp/probe` as the workspace directory.

   ```
   # Create a temporary directory to mount to the container
   $ mkdir /tmp/probe
   ```

3. Create a new file called `username.txt` and insert the username string into the file. For example:

   ```
   echo -n "<probe_username>" > /tmp/probe/username.txt
   ```

   Where `<probe_username>` is the username to authenticate with the ObjectServer
4. Verify that the following files are created in the temporary workspace directory.
   - `password.txt`
   - `username.txt`

   ```
   $ ls -1 /tmp/probe/
   password.txt
   username.txt
   ```

Extracting server certificates for SSL communication

For the probe to connect using SSL, the probe needs the server certificate in its key database in order to authenticate with the server when establishing a connection. The key database files are then put in a `Secret`. Follow the steps below to setup the key database files and add the server certificate.

This is section is only applicable if you are using either the `SSLOnly` or `SSLAndAuth` connection.

1. Contact your Netcool/OMNIbus Administrator to extract the SSL certificate from the server key database which will then be added into the key database of the probe client. Use the nc_gskcmd to extract the certificate from the server's key database on the server host, for example:

   ```
   $NCHOME/bin/nc_gskcmd -cert -extract \
   -db "$NCHOME/etc/security/keys/omni.kdb" \
   -pw password \
   -label "keylabel" \
   -target "$NCHOME/etc/security/keys/certname.arm"
   ```

   Where `password` is the password for the key database, `keylabel` is the description of the certificate in the key database (specify this value as a quoted string), and `certname` is the name of the certificate that you want to extract.

   Specify the path to the certificate as a quoted string. A sample command to extract the a root CA certificate with the label "`NCOMS_CA`" into the file `ca.crt`:

   ```

   $NCHOME/bin/nc_gskcmd -cert -extract \
   -db "$NCHOME/etc/security/keys/omni.kdb" \
   -pw password \
   -label "NCOMS_CA" \
   -target "$NCHOME/etc/security/keys/ca.crt"
   ```

2. Create a temporary workspace directory if necessary. The following steps uses `/tmp/probe` as the workspace directory.

   ```
   $ mkdir /tmp/probe
   ```

3. Copy the server certificate into the workspace directory.

```
$ cp ca.crt /tmp/probe
```

Create a probe-server communication secret

After preparing the required files, use `kubectl` to create a secret generic command with the `--from-file` option to create a Kubernetes secret.

Depending on the connection mode that is required by the ObjectServer, create a `Secret` with the required fields. The secret name should contain the helm release name as a prefix for easy reference.

Example commands are shown below for each connection mode using `probe` as the release name to create a secret called `probe-omni-secret` in the `noi-integrations` namespace.

- To create a secret for secured communication (`AuthOnly` connection mode):

```
kubectl create secret generic probe-omni-secret \
--namespace noi-integrations \
--from-file=AuthUserName=/tmp/probe/username.txt \
--from-file=AuthPassword=/tmp/probe/password.txt
```

- To create a secret for SSL enabled communication (`SSLOnly` connection mode):

```
kubectl create secret generic probe-omni-secret \
--namespace noi-integrations \
--from-file=ca.crt=ca.crt
```

- To create a secret for SSL enabled and secured communication (`SSLAndAuth` connection mode):

```
kubectl create secret generic probe-omni-secret \
--namespace noi-integrations \
--from-file=AuthUserName=/tmp/probe/username.txt \
--from-file=AuthPassword=/tmp/probe/password.txt \
--from-file=ca.crt=ca.crt
```

Note: Each field key must follow the expected names `AuthUserName`, `AuthPassword`, `tls.crt` and `ca.crt` are case sensitive. If any of the field key is misspelled, the pod might fail to mount to the secret to obtain the secret.

Remember to clean up this directory after you have successfully created the Kubernetes secret.

For Netcool Operations Insight (NOI) on RedHat Openshift Container Platform (OCP)

Preparing credentials for authentication

If you are connecting to an ObjectServer on RedHat Openshift Container Platform (OCP), you need to extract the CA certificate from the `Proxy` deployed with the NOI deployment and use the proxy to secure the connection using TLS. For more information on how to get the connection details and the proxy certificate, see Connecting with the proxy service.

Retrieve the ObjectServer authentication credentials:

```
 NOI_INSTANCE=<noi instance>
 NOI_NAMESPACE=<noi namespace>
 NOI_OMNI_USERNAME=root
 NOI_OMNI_PASSWORD=$(kubectl get secret -n $NOI_NAMESPACE $NOI_INSTANCE-omni-secret -o
jsonpath='{.data.OMNIBUS_ROOT_PASSWORD}' | base64 --decode && echo)
```

Where:

`<noi instance>` is the NOI instance name.

`<noi namespace>` is the NOI instance namespace.

Extracting server certificates for SSL communication

If the ObjectServer TLS certificate is not signed by a trusted CA or the Openshift Certificate Authority (CA) signer certificate, extract the TLS Certificate from the ObjectServer. The example below will extract the NOI proxy `tls.crt` file into the local directory.

```
NOI_INSTANCE=noi-evtmgr
NOI_NAMESPACE=noi
oc extract secret/$NOI_INSTANCE-proxy-tls-secret -n $NOI_NAMESPACE --to=. --keys=tls.crt
```

Create a probe-server communication secret

After the preparing the required files, use the `oc create secret generic` command with the `--from-file` option to create a Kubernetes secret.

Depending on the connection mode that is required by the ObjectServer, create a `Secret` with the required fields. The secret name should contain the helm release name as a prefix for easy reference.

Example commands are shown below for each connection mode using `probe` as the release name to create a secret called `probe-omni-secret` in the `noi-integrations` namespace.

- To create a secret for secured communication (`AuthOnly` connection mode):

  ```
  oc create secret generic probe-omni-secret \
  --namespace noi-integrations \
  --from-literal=AuthUserName=$NOI_OMNI_USERNAME \
  --from-literal=AuthPassword=$NOI_OMNI_PASSWORD
  ```

- To create a secret for SSL enabled communication (`SSLOnly` connection mode):

  ```
  oc create secret generic probe-omni-secret \
  --namespace noi-integrations \
  --from-file=tls.crt=tls.crt
  ```

- To create a secret for SSL enabled and secured communication (`SSLAndAuth` connection mode):

  ```
  oc create secret generic probe-omni-secret \
  --namespace noi-integrations \
  --from-literal=AuthUserName=$NOI_OMNI_USERNAME \
  --from-literal=AuthPassword=$NOI_OMNI_PASSWORD \
  --from-file=tls.crt=tls.crt
  ```

Note: Each field key must follow the expected names `AuthUserName`, `AuthPassword`, `tls.crt` and `ca.crt` are case sensitive. If any of the field key is misspelled, the pod might fail to mount to the secret to obtain the secret.

Remember to clean up this directory after you have successfully created the Kubernetes secret.

Securing Generic Webhook Probe and Event Source Communications

By default, the probe starts a secure internal webhook endpoint. To expose this webhook endpoint externally, you must enable Ingress (using OCP Route) by setting `ingress.enabled` to `true` in `deploy/crs/probes_v1_webhookprobe.yaml`. The example below creates a Route with the default OCP service serving certificate to secure the ingress.

```
ingress:
  enabled: true
  # "host" is an alias/DNS that points to the service and optional. If not specified, a route
host will typically be automatically chosen.
  # You may override this parameter and it must follow DNS952 subdomain conventions.
  # Run oc get ingresses.config/cluster -o jsonpath={.spec.domain} to get default ingress
domain.
  host: ""

# Transport (Webhook) specified configuration
webhook:
  uri: /probe
  # HTTP Versions supported are 1.1 or 1.0
```

```
  httpVersion: "1.1"
  respondWithContent: "OFF"
  validateBodySyntax: "ON"
  validateRequestURI: "ON"
  idleTimeout: 180

  tls:
    enabled: true
    # Pre-created secret contains custom tls.crt and tls.key to encrypt the communications.
    # If unset, the probe will use default OCP service certificate to encrypt the 
communications.
    secretName: ""

  # A pre-created secret contains serverBasicAuthenticationUsername and 
serverBasicAuthenticationPassword
  # to specify the webhook (server) basic authentication username and password
  serverBasicAuthenticationCredentialsSecretName: ""

  replicaCount: 1

  autoscaling:
    enabled: false
    minReplicas: 1
    maxReplicas: 3
    cpuUtil: 60
```

To secure Ingress with a custom TLS private key (`tls.key`) and certificate (`tls.crt`), you must pre-create a secret that contains the custom `tls.key` and `tls.crt`. To create a simple self-signed certificate using OpenSSL, run the following command:

```
openssl req -x509 -newkey rsa:4096 -keyout tls.key -out tls.crt -days 365 -nodes
```

The example below creates the secret `ingress-custom-tls` that contains custom the TLS private key (`tls.key`) and certificate (`tls.crt`).

```
oc create secret tls ingress-custom-tls \
  --key=<path to tls.key> \
  --cert=<path to tls.crt>
```

Then set `webhook.tls.secretName` to `ingress-custom-tls` as in the example below to allow the probe to use the custom TLS certificate for webhook connection.

```
ingress:
  enabled: true
  # "host" is an alias/DNS that points to the service and optional. If not specified, a route 
host will typically be automatically chosen.
  # You may override this parameter and it must follow DNS952 subdomain conventions.
  # Run oc get ingresses.config/cluster -o jsonpath={.spec.domain} to get default ingress 
domain.
  host: ""

# Transport (Webhook) specified configuration
webhook:
  uri: /probe
  # HTTP Versions supported are 1.1 or 1.0
  httpVersion: "1.1"
  respondWithContent: "OFF"
  validateBodySyntax: "ON"
  validateRequestURI: "ON"
  idleTimeout: 180

  tls:
    enabled: true
    # Pre-created secret contains custom tls.crt and tls.key to encrypt the communications.
    # If unset, the probe will use default OCP service certificate to encrypt the 
communications.
    secretName: "ingress-custom-tls"

  # A pre-created secret contains serverBasicAuthenticationUsername and 
serverBasicAuthenticationPassword
  # to specify the webhook (server) basic authentication username and password
  serverBasicAuthenticationCredentialsSecretName: ""

  replicaCount: 1

  autoscaling:
    enabled: false
    minReplicas: 1
    maxReplicas: 3
    cpuUtil: 60
```

Sending HTTP Requests to Event Source

To configure the HTTP client component of the probe to send HTTP requests to the event source, you must pre-create a secret that contains the `httpRequests` data key. `httpRequests` contains the list of HTTP request configurable parameters shown below. You must update related parameters and remove unrequired entries then save them in a local JSON file.

```
{
   "GLOBAL":
   {
      "httpVersion":"1.1",
      "httpHeaders":"",
      "responseTimeout":"60",
      "securityProtocol":"",
      "keepTokens":"",
      "tokenEndpointURI":"",
      "autoReconnect":"OFF"
   },

   "LOGIN":
   {
        "GET_ACCESS_TOKEN":
        {
            "uri":"",
            "method":"POST",
            "headers":"",
            "content":"",
            "interval":"0",
            "attempts":"0",
            "requireSSL":"true",
            "enableHostnameVerification" : "true"
        },

        "GET_REFRESH_ACCESS_TOKEN":
        {
            "uri":"",
            "method":"POST",
            "headers":"",
            "content":"",
            "interval":"60",
            "attempts":"0",
            "requireSSL":"true",
            "enableHostnameVerification" : "true"
        }
   },

   "SUBSCRIBE":
   {
        "GET_SUBSCRIPTION":
        {
            "uri":"",
            "method":"POST",
            "headers":"",
            "content":"",
            "interval":"0",
            "attempts":"0",
            "requireSSL":"true",
            "enableHostnameVerification" : "true"
        },

        "GET_SUBSCRIPTION_REFRESH":
        {
            "uri":"",
            "method":"POST",
            "headers":"",
            "content":"",
            "interval":"120",
            "attempts":"0",
            "requireSSL":"true",
            "enableHostnameVerification" : "true"
        }
   },
   "RESYNC":
   {
        "RESYNC_CHILD_BY_FDN":
        {
            "uri":"",
            "method":"POST",
            "headers":"",
            "content":"",
            "interval":"0",
            "attempts":"0",
            "requireSSL":"true",
            "enableHostnameVerification" : "true"
        }
   },

   "LOGOUT":
   {
        "REVOKE_ACCESS_TOKEN":
        {
            "uri":"",
            "method":"POST",
            "headers":"",
            "content":"",
            "interval":"0",
            "attempts":"0",
            "requireSSL":"true",
            "enableHostnameVerification" : "true"
        }
   }
}
```

The example below shows the HTTP request parameters in JSON file to allow the probe to send HTTP requests to `https://<target host>/api/notifications/pull` and retrieve `Location` field in HTTP response from the server. The `Location` field contains the event source URI and it will be saved as token that will be used for subsequent HTTP requests to perform resynchronization.

```
{
    "GLOBAL": {
        "httpVersion": "1.1",
        "httpHeaders": "Accept=application/json,Content-Type=application/json,Use-
Cookie=true,User-Agent=IBM Netcool/OMNIBus Message Bus Probe",
        "responseTimeout": "60",
        "securityProtocol": "TLSv1.2",
        "keepTokens": "Location",
        "tokenEndpointURI": "",
        "autoReconnect": "OFF"
    },
    "LOGIN": {
        "GET_ACCESS_TOKEN": {
            "uri": "https://<target host>/api/notifications/pull",
            "method": "POST",
            "headers": "",
            "content": "",
            "interval": "0",
            "requireSSL": "true",
            "enableHostnameVerification" : "true"
        }
    },
    "RESYNC": {
        "RESYNC_FAULT_MANAGEMENT_ALARMS": {
            "uri": "++Location++",
            "method": "GET",
            "headers": "",
            "content": "",
            "interval": "0",
            "requireSSL": "true",
            "enablePagination": "false",
            "enableHostnameVerification" : "true"
        }
    }
}
```

The following table shows descriptions of each parameter in the example above.

Parameter name	Description
GLOBAL.httpVersion	Use this property to specify the version of the HTTP protocol that the target system supports. In the example, it is set to `1.1`.
GLOBAL.httpHeaders	Use this property to specify the HTTP header options to use in all HTTP requests. In the example, it is set to `Accept=application/json,Content-Type=application/json,Use-Cookie=true,User-Agent=IBM Netcool/OMNIBus Message Bus Probe`.
GLOBAL.responseTimeout	Use this property to specify how long (in seconds) theprobe waits for a response from the target system before timing out. In the example, it is set to `60` seconds.
GLOBAL.securityProtocol	Use this property to specify the security protocol to use when retrieving events from the REST API. In the example, it is set to `TLSv1.2`.
GLOBAL.keepTokens	Use this property to specify a comma-separated list of the attributes that the probe extracts from the incoming JSON data. These data items can be used in token substitution throughout the runtime of the probe. In the example, it is set to `Location`. `Location` will be extracted from incoming JSON data during login. `Location` contains the URI of the event source.
GLOBAL.tokenEndpointURI	Use this property to specify the URI that the probe uses to obtain an access token for the target device. This is the path on the remote host to request an access token, for example: `tokenEndpointURI=/oauth/token`. In the example, it is set to `nil`.
GLOBAL.autoReconnect	Use this property to switch ON or OFF to perform auto-reconnection. In the example, it is set to `OFF`.
LOGIN.GET_ACCESS_TOKEN.uri	Use this property to specify the URI that the probe uses to request a login. In the example, it is set to `https://<target host>/api/notifications/pull`.
LOGIN.GET_ACCESS_TOKEN.method	Use this property to specify the message type that the probe sends to request a login. In the example, it is set to `POST` to post the HTTP request.
LOGIN.GET_ACCESS_TOKEN.headers	Use this property to specify an HTTP header to send with all login requests. This overrides the global HTTP header options configured by the **GLOBAL.httpHeaders** property. In the example, it is set to `nil`.

Parameter name	Description
LOGIN.GET_ACCESS_TOKEN.content	Use this property to specify any additional information that the probe sends with the login request. In the example, it is set to `nil`.
LOGIN.GET_ACCESS_TOKEN.interval	Use this property to specify the interval (in seconds) that the probe leaves between successive login. In the example, it is set to `0` to disable this feature.
LOGIN.GET_ACCESS_TOKEN.requireSSL	Use this property to enable or disable TLS/SSL connection during login. In the example, it is set to `true` to enable TLS/SSL connection.
LOGIN.GET_ACCESS_TOKEN.enableHostnameVerification	Use this property to enable or disable hostname verification during login. Hostname verification requires that the hostname of the target host must match the Subject Alternate Name (SAN) or DNS entry of the provided server certificate. If there is no match, an exception is thrown and the TLS handshake is aborted. This is to avoid man-in-the-middle attack which may present a different TLS server cert from the target hostname the HTTP client is trying to connect to. In the example, it is set to `true` to enable hostname verification during login.
RESYNC.RESYNC_FAULT_MANAGEMENT_ALARMS.uri	Use this property to specify the URI that the probe uses to request a resynchronization with the target system. In the example, it is set to `++Location++` to use the token specified in **GLOBAL.keepTokens**. `++Location++` contains the event source URI.
RESYNC.RESYNC_FAULT_MANAGEMENT_ALARMS.method	Use this property to specify the message type that the probe sends to request a resynchronization with the target system. In the example, it is set to GET to get events from event source during resynchronization.
RESYNC.RESYNC_FAULT_MANAGEMENT_ALARMS.headers	Use this property to specify an HTTP header to send with all resynchronization requests. This overrides the global HTTP header options configured by the **GLOBAL.httpHeaders** property. In the example, it is set to `nil`.
RESYNC.RESYNC_FAULT_MANAGEMENT_ALARMS.content	Use this property to specify any additional information that the probe sends with the resynchronization request. In the example, it is set to `nil`.
RESYNC.RESYNC_FAULT_MANAGEMENT_ALARMS.interval	Use this property to specify the interval (in seconds) that the probe leaves between successive login. In the example, it is set to `0` to disable this feature.

Parameter name	Description
RESYNC.RESYNC_FAULT_MANAGEMENT_ALARMS.requireSSL	Use this property to enable or disable TLS/SSL connections during resynchronization. In the example, it is set to `true` to enable TLS/SSL connections.
RESYNC.RESYNC_FAULT_MANAGEMENT_ALARMS.enablePagination	Use this property to enable or disable pagination. In the example, it is set to `false` to disable pagination.
RESYNC.RESYNC_FAULT_MANAGEMENT_ALARMS.enableHostnameVerification	Use this property to enable or disable hostname verification during resync. Hostname verification requires that the hostname of the target host must match the Subject Alternate Name (SAN) or DNS entry of the provided server certificate. If there is no match, an exception is thrown and the TLS handshake is aborted. This is to avoid man-in-the-middle attack which may present a different TLS server cert from the target hostname the HTTP client is trying to connect to. In the example, it is set to `true` to enable hostname verification during resync.

The following command creates a new secret named `http-requests` that contains `httpRequests` data with the HTTP request parameters saved in `restWebhookTransport.properties` file.

```
oc create secret generic http-requests \
  --from-file=httpRequests=<path to restWebhookTransport.properties>
```

Then set `httpClient.httpRequestsSecretName` to `http-requests` as in the example below to allow the probe to send HTTP requests to event source.

```
# (Optional) HTTP Client Configuration
httpClient:
  ## Set to 'true' to deploy HTTP Client component. Default is 'false'.
  enabled: true

  ## Host and Port parameters are used by the HTTP client component to connect
  ## to an event source.
  host: ""
  port: 80

  ## Pre-created secret contains server.crt of an event source which is TLS/SSL enabled.
  ## To allow probe to connect to the event source
  sslSecretName: ""

  ## Pre-created secret contains HTTP Request Configuration
  ## Refer to README for details
  httpRequestsSecretName: "http-requests"
```

Sending Secure HTTP Requests to Event Resource

In addition to the steps described in "Sending HTTP Requests to Event Source" on page 125, to configure the HTTP client component of the probe, you must pre-create a secret that contains the `server.crt` data key to secure the connection. `server.crt` contains event source TLS/SSL certificate(s). The command below creates a secret named `probe-client-tls-cert` that contains the `server.crt` data key.

```
oc create secret generic probe-client-tls-cert \
  --from-file=server.crt=<path to event source certificate file>
```

You can insert multiple event source certificate files into a certificate file. The following command concatenates multiple certificate files into a certificate file which will be used by the secret creation command above.

```
cat event-source-1-certificate.pem event-source-2-certificate.pem > server-certificate.pem
```

Then set `httpClient.sslSecretName` to `probe-client-tls-cert` as in the example below to allow the probe to send HTTP requests to the TLS/SSL enabled event source.

```
# (Optional) HTTP Client Configuration
httpClient:
  ## Set to 'true' to deploy HTTP Client component. Default is 'false'.
  enabled: true

  ## Host and Port parameters are used by the HTTP client component to connect
  ## to an event source.
  host: ""
  port: 80

  ## Pre-created secret contains server.crt of an event source which is TLS/SSL enabled.
  ## To allow probe to connect to the event source
  sslSecretName: "probe-client-tls-cert"

  ## Pre-created secret contains HTTP Request Configuration
  ## Refer to README for details
  httpRequestsSecretName: "http-requests"
```

Deploying the Generic Webhook Probe

To deploy a new instance of the Generic Webhook Probe, see to the table below for the descriptions of each configurable parameter. See also "HTTP server (Webhook) configuration" on page 136 and "HTTP client configuration" on page 138.

Then, update `deploy/crs/probes_v1_webhookprobe.yaml` accordingly and save the YAML file.

Parameter name	Description
spec.license.accept	The license state of the image being deployed. Enter `true` to install and use the image. The default value is `false`.
spec.version	Probe application version. Accepted versions are `3.1.0`, `3.2.0`, or `3.3.0`. The default is `3.3.0`.
spec.helmValues.global.image.secretName	Name of the Secret containing the Docker Config to pull the image from a private repository. Leave blank if the probe image already exists in the local image repository or if the Service Account has already been assigned with an Image Pull Secret. The default is `nil`.
spec.helmValues.global.serviceAccountName	Name of the service account to be used by the probe instance. If the Cluster Administrator has already created a service account in the namespace, specify the name of the service account here. If left blank, the operator will automatically create a new service account in the namespace when it is deployed. This new service account will be removed from the namespace when the instance is removed. The default is `nil`.

Parameter name	Description
spec.helmValues.global.enableNetworkPolicy	Enables NetworkPolicy to allow traffic from and to the pod. The default is `true`.
spec.helmValues.image.probe	Probe image repository. The image name must be `netcool-probe-messagebus`. The default is `cp.icr.io/cp/noi-int/netcool-probe-messagebus@sha256:7d524c2bebd30640bbbc2ed89367800b65b6cfdc2c53a1e5785577284af36956`.
spec.helmValues.image.utility	The Utility image repository. The image name must be `netcool-integration-util`. The default is `cp.icr.io/cp/noi-int/netcool-integration-util@sha256:97beb12f8660cdc2392507e9e0d42c026b7ff41f3a5c811785621af8a3906e0a`.
spec.helmValues.image.pullPolicy	Image pull policy. The default is `IfNotPresent`.
spec.helmValues.netcool.connectionMode	The connection mode to use when connecting to the Netcool/OMNIbus ObjectServer. Refer to Securing Probe and ObjectServer Communications section for more details. **Note**: Refer to limitations section for more details on available connection modes for your environment. The default is `default`.
spec.helmValues.netcool.primaryServer	The primary Netcool/OMNIbus server the probe should connect to (required). Usually set to `NCOMS` or `AGG_P`. The default is `nil`.
spec.helmValues.netcool.primaryHost	The host of the primary Netcool/OMNIbus ObjectServer (required). Specify the ObjectServer Hostname or IP address. The default is nil.
spec.helmValues.netcool.primaryPort	The port number of the primary Netcool/OMNIbus server (required). The default is nil.
spec.helmValues.netcool.backupServer	The backup Netcool/OMNIbus ObjectServer to connect to. If the **backupServer**, **backupHost** and **backupPort** parameters are defined in addition to the **primaryServer**, **primaryHost**, and **primaryPort** parameters, the probe will be configured to connect to a virtual ObjectServer pair called `AGG_V`. The default is `nil`.
spec.helmValues.netcool.backupHost	The host of the backup Netcool/OMNIbus ObjectServer. Specify the ObjectServer Hostname or IP address. The default is `nil`.
spec.helmValues.netcool.backupPort	The port of the backup Netcool/OMNIbus ObjectServer. The default is `nil`.

Parameter name	Description
spec.helmValues.netcool.secretName	A pre-created secret for `AuthOnly`, `SSLOnly` or `SSLAndAuth` connection mode. Certain fields are required depending on the connection mode. The default is `nil`.
spec.helmValues.netcool.waitTimeout	The duration (in minutes) to wait for the ObjectServer to become available. The default is `0`, disables this feature.
spec.helmValues.probe.messageLevel	Probe log message level. The default is `warn`.
spec.helmValues.probe.setUIDandGID	When set to `true`, pods are run with specific UID and GID values. Otherwise, when set to `false` the pods run with an arbitrary UID, for example when deploying in the OpenShift native restricted `SecurityContextConstraints`. The default is `false`. When set to `true`, the operator will specify the UID and GID values for the netcool user else the netcool user will not be assigned any UID or GID by the operator. When deploying in an environment that is enforced by SecurityContextConstraints (SCC) that requires the process to have a fixed UID and GID value, this parameter must be set to `true`. Otherwise, if deploying in an environment enforced by an SCC that requires the process to support arbitrary UIDs, for example in the Openshift `restricted` SCC, this parameter must be set to `false`. Default value is `false`.
spec.helmValues.probe.sslServerCommonName	A comma-separated list of acceptable SSL `Common Names` when connecting to ObjectServer using SSL. This should be set when the **CommonName** field of the received certificate does not match the name specified by the **primaryServer** property. When a **backupServer** is specified, the probe will create an ObjectServer pair with `AGG_V` as the name. Set this parameter if the `Common Name` of the certificate does not match `AGG_V`. The default is `nil`.
spec.helmValues.probe.locale	Probe environment locale setting. Used as the `LC_ALL` environment variable. The default is `en_US.utf8`.
spec.helmValues.probe.enableTransportDebugLog	Set to `true` to enable common Transport Module debug logging for troubleshooting purposes. Default value is `false`.
spec.helmValues.probe.heartbeatInterval	Specifies the frequency (in seconds) with which the probe checks the status of the host server. The default is `10`.

Parameter name	Description
spec.helmValues.probe.recordData	(Optional) A comma-separated list of attributes from the event to be recorded in the probe data backup file. The data recorded with the prefix `RecordData.` can be used by the probe to resolve transport properties using tokens with a prefix and suffix of ++. For example, `date_happened` is an event attribute and it can be resolved by the probe if specified as `"uri":"https://<event source host>:443/api/v1/events?start=++date_happened++&end=++now++&page=0"` in the transport properties file. The probe will substitute `++date_happened++` with the value of `RecordData.date_happened` stored in the backup file. The default is `nil`
spec.helmValues.probe.configPVC.name	(Optional) Specifies a pre-created Persistent Volume Claim name to access custom probe files in Persistent Volume. (Optional) Specifies a pre-created Persistent Volume Claim name to access custom probe files in Persistent Volume. If left unset, the probe will use the default Message Bus probe files. The default is `nil`
spec.helmValues.probe.configPVC.rulesFile	(Optional) Specifies the path to the main probe rules file. If left unset, the probe will use the default Message Bus probe rules files. This field will override the selection made for the `probe.rulesConfigmap` parameter. The default is `nil`
spec.helmValues.probe.configPVC.parserConfigFile	(Optional) Specifies the path to the JSON parser configuration file. If left unset, the probe will use the default JSON parser configuration. This field will override the selection made for the `spec.helmValues.probe.jsonParserConfig.messagePayload`, `spec.helmValues.probe.jsonParserConfig.messageHeader`, `spec.helmValues.probe.jsonParserConfig.jsonNestedPayload`, `spec.helmValues.probe.jsonParserConfig.jsonNestedHeader` and `spec.helmValues.probe.jsonParserConfig.messageDepth` parameters. The default is `nil`.
spec.helmValues.probe.configPVC.dataBackupFile	(Optional) Specifies the path to probe data backup file. If left unset, the probe will store probe backup data in default probe data backup file. The default is `nil`.
spec.helmValues.probe.rulesConfigmap	(Optional) Specifies a customized `ConfigMap` to be used for the rules files. This field must be left unset if not required. The default is `nil`.

Parameter name	Description
spec.helmValues.probe.jsonParserConfig.notification.messagePayload	Specifies the JSON tree to be identified as message payload from the notification (webhook) channel. See example for more details on how to configure the Probe's JSON parser. The default is `json`.
spec.helmValues.probe.jsonParserConfig.notification.messageHeader	(Optional) Specifies the JSON tree to be identified as message header from the notification (webhook) channel. Attributes from the headers will be added to the generated event. The default is `nil`.
spec.helmValues.probe.jsonParserConfig.notification.jsonNestedPayload	Specifies the JSON tree within a nested JSON or JSON string to be identified as message payload from the notification (webhook) channel. The `probe.jsonParserConfig.notification.messagePayload` parameter must be set to point to the attribute containing the JSON String. The default is `nil`.
spec.helmValues.probe.jsonParserConfig.notification.jsonNestedHeader	Specifies the JSON tree within a nested JSON or JSON string to be identified as message header from the notification (webhook) channel. The `probe.jsonParserConfig.notification.messageHeader` parameter must be set to point to the attribute containing the JSON String. The default is `nil`.
spec.helmValues.probe.jsonParserConfig.notification.messageDepth	Specifies the number of levels in the message to traverse during parsing. The default is 3.
spec.helmValues.probe.jsonParserConfig.resync.messagePayload	Specifies the JSON tree to be identified as message payload from the re-synchronization (HTTP REST API). See example for more details on how to configure the Probe's JSON parser. The default is `json`.
spec.helmValues.probe.jsonParserConfig.resync.messageHeader	Specifies the JSON tree to be identified as message header from the re-synchronization (HTTP REST API). Attributes from the headers will be added to the generated event. The default is `nil`.
spec.helmValues.probe.jsonParserConfig.resync.jsonNestedPayload	Specifies the JSON tree within a nested JSON or JSON string to be identified as message payload from the re-synchronization (HTTP REST API). The `probe.jsonParserConfig.resync.messagePayload` parameter must be set to point to the attribute containing the JSON String. The default is `nil`.

Parameter name	Description
spec.helmValues.probe.jsonParserConfig.resync.jsonNestedHeader	Specifies the JSON tree within a nested JSON or JSON string to be identified as message header from the re-synchronization (HTTP REST API). The `probe.jsonParserConfig.resync.messageHeader` parameter must be set to point to the attribute containing the JSON String. The default is `nil`.
spec.helmValues.probe.jsonParserConfig.resync.messageDepth	Specifies the number of levels in the message to traverse during parsing. The default is 3.
spec.helmValues.probe.httpClientCredentialsSecretName	The pre-created secret that contains the credentials (`Username` and `Password` key) that can be used in the HTTP client requests configuration to authenticate with the event source. The username and password values can be set in the HTTP client requests configuration using the `++Username++` and `++Password++` syntax. The default is `nil`.
spec.helmValues.probe.initialResync	Set to `true` to send a re-synchronization request before sending a subscription request. The default is `false`.
spec.helmValues.probe.resyncInterval	The interval (in seconds) to check the transport connection status. The default is `0`.
spec.helmValues.probe.integration	Specifies the integration type for the probe. Accepted values are `generic`, `sevone`, `turbonomic` or `datadog`. The default is `generic`. If set to `generic`, the probe is installed with a generic rules files and if set to a specific integration, the probe is installed with a custom rules for the integration.
spec.helmValues.probe.poddisruptionbudget.enabled	Set to `true` to enable Pod Disruption Budget to maintain high availability during a node maintenance. Administrator role or higher is required to enable Pod Disruption Budget on clusters with Role Based Access Control. The default is `false`.
spec.helmValues.probe.poddisruptionbudget.minAvailable	The number of minimum available pods during node drain. Can be set to a number or percentage, for example: 1 or 10%. Caution: Setting to 100% or equal to the number of replicas may block node drains entirely. The default is 1.
spec.helmValues.probe.selfMonitoring.discardHeartbeatEvent	Set this value to `true` if you want to discard the ProbeWatch Heartbeat event. Set this value to `false` if you want to forward the ProbeWatch Heartbeat event to the ObjectServer. The default is `false`.

Parameter name	Description
spec.helmValues.probe.selfMonitoring.populateMasterProbeStat	Set this value to `true` to enable a new probe metrics event to be generated by using the genevent function, which is defined within the case "Heartbeat ..." statement. The event data consists a set of OplStats probe metrics, which are forwarded to the `master.probestats` table that was created when you ran the `probestats.sql` script. Set this value to `false` if you do not want to generate this event for insertion into the `master.probestats` table. The default is `false`.
spec.helmValues.probe.selfMonitoring.logProbeStat	Set this value to `true` if you want to record the OplStats metrics in the probe log file. Details are logged at the INFO level. The metric details that are logged are defined in the case "Heartbeat ..." statement. Set this value to `false` if you do not want to record the metrics in the log file. The default is `false`.
spec.helmValues.probe.selfMonitoring.generateThresholdEvents	Set this value to `true` if you want to generate threshold events that indicate when a particular probe metric violates a defined threshold. By default, no code is provided for threshold events within this rules file because individual preferences can vary widely. If you require threshold events, you must first decide which thresholds you want to monitor. Then, within the case "Heartbeat ..." statement, provide the code for generating threshold events. The default is `false`.
spec.helmValues.probe.configs	A multiline string parameter to configure additional probe properties. Each entry must not contain space. The default is `ProbeWatchHeartbeatInterval:0 EventLoadProfiling:'true'`.
spec.helmValues.resources.limits.memory	Memory resource limits. The default is `256Mi`.
spec.helmValues.resources.limits.cpu	CPU resource limits. The default is `200m`.
spec.helmValues.resources.limits.ephemeral-storage	Ephemeral storage limits. The default is `200Mi`.
spec.helmValues.resources.requests.cpu	CPU resource requests. The default is `1000m`.
spec.helmValues.resources.requests.memory	Memory resource requests. The default is `128Mi`.
spec.helmValues.resources.requests.ephemeral-storage	Ephemeral storage requests. The default is `100Mi`.
spec.helmValues.arch	Worker node architecture. Fixed to `amd64`.

HTTP server (Webhook) configuration

The following table shows descriptions of the HTTP server component configuration parameters.

Parameter name	Description
spec.helmValues.ingress.enabled	Ingress enabled. The default is `false`.
spec.helmValues.ingress.host	(Optional) An alias/DNS that points to the service. If not specified, a route host will typically be automatically chosen. You may override this parameter and it must follow DNS952 subdomain conventions. You can run oc get ingresses.config/cluster -o jsonpath={.spec.domain} to retrieve default ingress domain. The default is `nil`.
spec.helmValues.webhook.uri	Probe's Webhook URI into which the target device will POST notifications. The default is `/probe`.
spec.helmValues.webhook.respondWithContent	Use this property to specify whether the probe includes the HTTP body received from the client HTTP request in the HTTP response. Set this property to `ON` to configure probe webhook to include the HTTP body. If set to `OFF_WITH_HTTP_200`, the probe will respond with `200 (OK)` for a successful request without a body in the response. The default is `OFF` and the probe will respond with `204 (No Content)` response code.
spec.helmValues.webhook.validateBodySyntax	Set to ON to perform a JSON format check on the HTTP request body. The default is `ON`.
spec.helmValues.webhook.validateRequestURI	Set to ON to enable URI path check. Setting this property to OFF disables the URI check and the webhook will accept all HTTP request regardless of the path set. The default is `ON`.
spec.helmValues.webhook.idleTimeout	The time (in seconds) to allow an idle HTTP client to be connected. The default is `180`.
spec.helmValues.webhook.tls.enabled	Set to true to enable TLS to encrypt communications with clients and target hosts. The default is `true`.
spec.helmValues.webhook.tls.secretName	The pre-created secret contains custom tls.crt and tls.key to be used in the TLS communication. If unset, the probe will use default OCP service serving certificate to encrypt the communications. The default is `nil`.
spec.helmValues.webhook.serverBasicAuthenticationCredentialsSecretName	The pre-created secret contains serverBasicAuthenticationUsername and serverBasicAuthenticationPassword to specify the webhook (server) basic authentication username and password. The default is `nil`.
spec.helmValues.webhook.replicaCount	Number of deployment replicas. This config is ignored when autoscaling.enabled set to true. The default is `1`.

Parameter name	Description
spec.helmValues.webhook.autoscaling.enabled	Set to true to enable auto-scaling. The default is `false`.
spec.helmValues.webhook.autoscaling.minReplicas	Minimum number of probe replicas. The default is `1`.
spec.helmValues.webhook.autoscaling.maxReplicas	Maximum number of probe replicas. The default is `3`.
spec.helmValues.webhook.autoscaling.cpuUtil	The target CPU utilization (in percentage). Example: 60 for 60% target utilization. The default is `60` .

HTTP client configuration

In addition to the "HTTP server (Webhook) configuration" on page 136, the probe can also be configured as a client to send HTTP requests such as to pull data (resynchronization) from a server.

The following table shows descriptions of the HTTP client component configuration parameters.

Parameter name	Description
spec.helmValues.httpClient.enabled	Set to true to deploy HTTP Client component. The default is `false`.
spec.helmValues.httpClient.host	The target server IP or hostname. The default is `nil`.
spec.helmValues.httpClient.port	The target server port. The default is `80`.
spec.helmValues.httpClient.sslSecretName	A pre-created secret contains server.crt. It is required when connecting to a HTTP server with TLS. The default is `nil`.
spec.helmValues.httpClient.httpRequestsSecretName	A pre-created secret contains HTTP request parameters. Refer to Configuring the WebHook transport for details. The default is `nil`.
spec.helmValues.httpClient.tokenEndpointURI	To specify the URI that the probe uses to obtain an access token for the target device. The default is `nil`.

Post-installation steps for the Generic Webhook Probe

Retrieving the probe webhook URL

Run the following command to determine the probe's `URI`, `ROUTE-ENABLED` status and `TLS-ENABLED` status.

```
PROBE_INSTANCE=<probe instance name>
oc get webhookprobe $PROBE_INSTANCE -o custom-columns=URI:.spec.helmValues.webhook.uri,ROUTE-
ENABLED:.spec.helmValues.ingress.enabled,TLS-ENABLED:.spec.helmValues.webhook.tls.enabled
```

If `ROUTE-ENABLED` returns `true`, then the `route` is enabled and exposed externally.

Run the following commands to retrieve the probe's webhook URL:

```
PROBE_NS=<probe namespace>
PROBE_HOSTNAME=$(oc get route --namespace $PROBE_NS $PROBE_INSTANCE-mb-webhook  -o
jsonpath="{.spec.host}")
PROBE_URI=$(oc get route --namespace $PROBE_NS $PROBE_INSTANCE-mb-webhook -o
jsonpath="{.spec.path}")
echo "Probe Webhook URL: https://$PROBE_HOSTNAME$PROBE_URI"
```

Note: You might need to map the hostname with the cluster IP address if it cannot be resolved by the DNS.

If `ROUTE-ENABLED` returns `false` and `TLS-ENABLED` returns `true`, the `route` is disabled and TLS communication is enabled.

Run the following commands to retrieve the probe's webhook URL:

```
PROBE_NS=<probe namespace>
PROBE_SERVICE=$(oc get service | grep $PROBE_INSTANCE-mb-webhook |  awk '{print $1}')
PROBE_URI=$(oc get webhookprobe $PROBE_INSTANCE -n $PROBE_NS --no-headers -o custom-
columns=URI:.spec.helmValues.webhook.uri)
PROBE_WEBHOOK_URL=https://$PROBE_SERVICE.$PROBE_NS.svc$PROBE_URI
echo "Probe Webhook URL: $PROBE_WEBHOOK_URL"
```

If `ROUTE-ENABLED` returns `false` and `TLS-ENABLED` returns `false`, the `route` is disabled and TLS communication is disabled.

Run the following commands to retrieve the probe's webhook URL:

```
PROBE_NS=<probe namespace>
PROBE_SERVICE=$(oc get service | grep $PROBE_INSTANCE-mb-webhook |  awk '{print $1}')
PROBE_URI=$(oc get webhookprobe $PROBE_INSTANCE -n $PROBE_NS --no-headers -o custom-
columns=URI:.spec.helmValues.webhook.uri)
PROBE_WEBHOOK_URL=http://$PROBE_SERVICE.$PROBE_NS$PROBE_URI
echo "Probe Webhook URL: $PROBE_WEBHOOK_URL"
```

Known issue: Probe unable to parse an array list

Currently the probe does not supported array list objects. You will see following errors messages in the probe log if you set `"gatherSubsTopicInfo":"true"` in the `Multi-channel HTTP Transport` properties configuration file and the response of the HTTP request returns the message in an array list object:

```
2022-03-31T02:01:25: Error: E-JPR-000-000: [assignTopFreshProp] Invalid state in the stack: the
holder object is an array list.
2022-03-31T02:01:25: Error: E-JPR-000-000: [parseFields] Exception from parsing Json fields:
[assignTopFreshProp] Invalid state in the stack: the holder object is an array list.
```

These errors indicate that the probe failed to parse the HTTP response in an array list to gather the subtopic information.

Workaround

Modify the HTTP response to be sent as another valid JSON object so that probe able to parse it.

Common pre-installation tasks for the IBM SevOne, IBM Turbonomic, and Datadog probes

The IBM SevOne and IBM Turbonomics probes share two pre-installation tasks:

- Installing the IBM Netcool Operations Insight Event Integrations Operator
- Gathering the ObjectServer connection information

Installing the IBM Netcool Event Integrations Operator

Install the IBM Netcool Operations Insight Event Integrations Operator. Follow the instructions in "Installing the IBM Netcool Operations Insight Event Integrations Operator" on page 21 or "Installing the IBM Netcool Operations Insight Event Integrations Operator to a private container registry” on page 26.

Gathering ObjectServer connection information

You need to gather several pieces of information from the ObjectServer before configuring the probe so that the probe can connect to the ObjectServer service.

The following information is required:

- Primary and backup ObjectServer service name and port
- ObjectServer client credentials
- TLS certificate used by the ObjectServer

1. Determine the ObjectServer instance name.

   ```
   kubectl get noi --all-namespaces
   ```

   Take note of the NAME and NAMESPACE from the output of the command above. This is the NOI instance name and namespace.
2. Run the commands below to gather the required service information and credentials.

   ```
    NOI_INSTANCE=<noi instance>
    NOI_NAMESPACE=<noi namespace>
    NOI_PROXY_SVC=$NOI_INSTANCE-proxy
    NOI_PROXY_PRIMARY_PORT=$(kubectl get svc -n $NOI_NAMESPACE $NOI_PROXY_SVC -o
   jsonpath='{.spec.ports[?(@.name=="aggp-proxy-port")].port}')
    NOI_PROXY_BACKUP_PORT=$(kubectl get svc -n $NOI_NAMESPACE $NOI_PROXY_SVC -o
   jsonpath='{.spec.ports[?(@.name=="aggb-proxy-port")].port}')
    NOI_OMNI_USERNAME=root
    NOI_OMNI_PASSWORD=$(kubectl get secret -n $NOI_NAMESPACE $NOI_INSTANCE-omni-secret -o
   jsonpath='{.data.OMNIBUS_ROOT_PASSWORD}' | base64 --decode && echo)
   ```

   Where:

   - `<noi instance>` is the NOI instance name.
   - `<noi namespace>` is the NOI instance namespace.
3. (Optional) If the ObjectServer TLS certificate is not signed by a trusted CA or the Openshift Certificate Authority (CA) signer certificate, extract the TLS Certificate from the Object Server. This certificate will be used in the probe secret in the following steps.

   ```
   oc extract secret/$NOI_INSTANCE-proxy-tls-secret -n $NOI_NAMESPACE --to=. --keys=tls.crt
   ```
4. Proceed to install the probe.

Deploying the Probe for IBM SevOne NPM Integration

IBM Netcool/OMNIbus Probe for IBM SevOne NPM creates the webhook endpoints to receive notification in the form of HTTP POST requests from IBM SevOne NPM.

Note: Since the release of IBM Cloud Pak AIOps 4.1, you can now use the Generic Webhook Connector to integrate IBM Cloud Pak for AIOps with IBM SevOne Network Performance Management (NPM) to digest events. Using the Webhook Connector is preferred over installing a probe to connect with IBM SevOne as it simplifies the setup process. For details seeCreating Generic Webhook connections

Note: The IBM Netcool Operations Insight Event Integrations Operator must be deployed before deploying this integration. See “Installing the IBM Netcool Operations Insight Event Integrations Operator” on page 21.

To deploy a new instance of the Generic Webhook Probe, review the custom resource YAML template supplied with the operator package, update the parameters if required, and save the configurable parameters in a file, for example `probes_v1_webhookprobe_sevone_integration.yaml`.

Then deploy the custom resource using the updated file:

```
kubectl apply -f deploy/crs/probes_v1_webhookprobe_sevone_integration.yaml --namespace
<NAMESPACE>
```

Note: If you are installing using the CLI, you can get the custom resource Yaml file from the `ibm-netcool-integrations/inventory/netcoolIntegrationsOperatorSetup/files/op-cli/deploy/crs` directory in our CASE archive. If you are installing using the Operator Lifecycle Manager (OLM) Web console, the custom resource is provided in the YAML view.

Prerequisites

The following systems is a prerequisite for this integration.

1. IBM Cloud Pak for AIOps 3.4 or IBM Netcool Operations Insight (NOI) 1.6.5 in Openshift Container Platform.

   Note: The Probe for IBM SevOne NPM Integration integration is provided as a Technology Preview for IBM Netcool Operations Insight (NOI) 1.6.5.

   However, the integration is available for production and support under the IBM Cloud Pak for AIOps 3.4 offering. See the following:

   - Announcement Letter of IBM Cloud Pak for AIOps 3.4
   - Announcement Letter of IBM Netcool Operations Insight V1.6
2. IBM SevOne NPM (IBM SevOne NMS 6.1).

Pre-installation tasks

The Probe for SevOne, Probe for Turbonomics, and Probe for Datadog have the same pre-installation requirements. See "Common pre-installation tasks for the IBM SevOne, IBM Turbonomic, and Datadog probes" on page 139 before proceeding with the following sections.

Configuring and installing the Probe for SevOne integration

Follow the instructions below to create the required secrets, configure the `WebhookProbe` custom resource and install the probe in the namespace.

1. Create a secret with the ObjectServer credentials for the probe to authenticate with the ObjectServer.

   ```
   PROBE_OMNI_SECRET=noi-probe-secret
   kubectl create secret generic $PROBE_OMNI_SECRET --from-
   literal=AuthUserName=$NOI_OMNI_USERNAME --from-literal=AuthPassword=$NOI_OMNI_PASSWORD --
   from-file=tls.crt=tls.crt
   ```
2. Create a secret for the probe to use for basic authentication.

   ```
   PROBE_AUTH_SECRET=sevone-probe-client-basic-auth
   kubectl create secret generic
   $PROBE_AUTH_SECRET --from-literal=serverBasicAuthenticationUsername=<username>  --from-
   literal=serverBasicAuthenticationPassword=<password>
   ```

   Where `<username>` is a user name and `<password>` is the password for SevOne NMS to use as basic authentication.
3. Create a Network Policy in the NOI namespace to allow the probe to access the TLS Proxy service and port in NOI namespace. **Note**: Review any other Network Policy that may be denying access to the ObjectServer TLS Proxy pod and update the policy to allow ingress connection to the TLS Proxy pod.

   ```
   cat << EOF | tee >(kubectl apply -f -) | cat
   kind: NetworkPolicy
   apiVersion: networking.k8s.io/v1
   metadata:
     name: noi-allow-proxy
     namespace: ${NOI_NAMESPACE}
   spec:
     podSelector:
   ```

```
    matchLabels:
      app.kubernetes.io/instance: ${NOI_INSTANCE}
      app.kubernetes.io/name: proxy
  ingress:
    - ports:
        - protocol: TCP
          port: ${NOI_PROXY_PRIMARY_PORT}
        - protocol: TCP
          port: ${NOI_PROXY_BACKUP_PORT}
  policyTypes:
    - Ingress
EOF
```

4. Create a Probe for SevOne Integrations with the `WebhookProbe` custom resource by running the commands below.

```
PROBE_SEVONE_INSTANCE=sevone-probe

cat << EOF | tee >(kubectl apply -f -) | cat
apiVersion: probes.integrations.noi.ibm.com/v1
kind: WebhookProbe
metadata:
  name: ${PROBE_SEVONE_INSTANCE}
  labels:
    app.kubernetes.io/name: ${PROBE_SEVONE_INSTANCE}
    app.kubernetes.io/managed-by: netcool-integrations-operator
    app.kubernetes.io/instance: ${PROBE_SEVONE_INSTANCE}
  namespace: ${NAMESPACE}
spec:
  helmValues:
    netcool:
      backupHost: '${NOI_PROXY_SVC}.${NOI_NAMESPACE}.svc'
      backupPort: ${NOI_PROXY_BACKUP_PORT}
      backupServer: 'AGGB'
      connectionMode: SSLAndAuth
      primaryHost: '${NOI_PROXY_SVC}.${NOI_NAMESPACE}.svc'
      primaryPort: ${NOI_PROXY_PRIMARY_PORT}
      primaryServer: 'AGGP'
      secretName: '${PROBE_OMNI_SECRET}'
    probe:
      jsonParserConfig:
        notification:
          jsonNestedHeader: ''
          jsonNestedPayload: ''
          messageDepth: 3
          messageHeader: ''
          messagePayload: json
      integration: sevone
      enableTransportDebugLog: false
      messageLevel: debug
    ingress:
      enabled: true
      host: ''
    arch: amd64
    webhook:
      uri: /probe/sevone
      serverBasicAuthenticationCredentialsSecretName: '${PROBE_AUTH_SECRET}'
      tls:
        enabled: true
        secretName: ''
  license:
    accept: true
  version: 3.1.0
EOF
```

5. Verify that the probe pod is running.

```
kubectl get pods -l app.kubernetes.io/instance=$PROBE_SEVONE_INSTANCE
```

Installing the Gateway

To integrate with IBM Cloud Pak for AIOps, follow the instructions in Installing IBM Cloud Pak for AIOps to setup the event flow into IBM Cloud Pak for AIOps. You may skip this step if there is already a gateway instance installed.

Obtaining the Probe Webhook URL

Use the following command to get the probe webhook URL which will be used to configure a Webhook destination in IBM SevOne NMS.

```
PROBE_HOSTNAME=$(oc get route $PROBE_SEVONE_INSTANCE-mb-webhook -o jsonpath='{.spec.host}')
PROBE_URI=$(oc get route $PROBE_SEVONE_INSTANCE-mb-webhook -o jsonpath='{.spec.path}')
PROBE_WEBHOOK_URL=https://$PROBE_HOSTNAME$PROBE_URI
echo"$PROBE_WEBHOOK_URL"
```

Configuring IBM SevOne NMS to forward events to the Probe for SevOne

IBM SevOne NMS must be configured with a new webhook configuration to send alerts to the Probe for SevOne NMS. **Note** IBM SevOne requires the webhook destination to use a TLS certificate signed by a trusted Certificate Authority (CA).

1. Login to SevOne console.
2. On the top menu, go to Events > Configuration > Policy Browser
3. On the Policy Browser page, click on **Configure Webhook Destination** button on the Policies panel.
4. On the Webhook Destination Manager page, click on **Add Webhook Destination** button.
5. On the Add Webhook Destination panel:
   a. Set a Webhook Destination Name, for example: `CP4WAIOps Probe for SevOne`
   b. Set a Description, for example: IBM CloudPak for WAIOps - Probe for SevOne NMS
   c. Set the URL with the `$PROBE_WEBHOOK_URL` value from the **Obtaining the Probe Webhook URL** section above.
   d. Check the "Use Basic Auth" option and set the **Username** and **Password** fields with the credentials set in the `PROBE_AUTH_SECRET` from the **Configuring and installing the Probe for SevOne integration** section above.
6. Click **Save**.
7. Configure policies to send events to probe by selecting the probe as the webhook destination. In each of the policies, configure the Webhook with the following details:
   a. **Method** set to `POST`
   b. **Content-Type** set to `application/json`
   c. **Webhook Destination** set to `CP4WAIOps Probe for SevOne`

      Note: This should be name that you specified for the webhook destination in Step 5.
   d. **Body** set with the following template for "Trigger Conditions"

```
{
"routes": [ "Netcool" ],
"host" : "$deviceName",
"description": "$alertMessage",
"alertMessage": "$alertMessage",
"check": "$policyName - $objectName",
"cluster": "$groupName",
"alertId": "$alertId",
"alertType": "$alertType",
"alertState": "$alertState",
"occurrences": "$occurrences",
"assignedTo": "$assignedTo",
"deviceId": "$deviceId",
"deviceIp": "$deviceIp",
"deviceName": "$deviceName",
"deviceAltName": "$deviceAltName",
"groupName": "$groupName",
"objectId": "$objectId",
"objectName": "$objectName",
"objectAltName": "$objectAltName",
"objectDescription": "$objectDescription",
"pluginName": "$pluginName",
"pluginDescription": "$pluginDescription",
"policyId": "$policyId",
"policyName": "$policyName",
"thresholdId": "$thresholdId",
"thresholdName": "$thresholdName",
"triggeringConditions": $triggeringConditions
}
```

For the "Clear Conditions", use the following template. This template sets the `alertState` attribute to `Cleared` as clear event.

```
{
"routes": [ "Netcool" ],
"host" : "$deviceName",
"description": "$alertMessage",
"alertMessage": "$alertMessage",
"check": "$policyName - $objectName",
"cluster": "$groupName",
"alertId": "$alertId",
"alertType": "$alertType",
"alertState": "Cleared",
"occurrences": "$occurrences",
"assignedTo": "$assignedTo",
"deviceId": "$deviceId",
"deviceIp": "$deviceIp",
"deviceName": "$deviceName",
"deviceAltName": "$deviceAltName",
"groupName": "$groupName",
"objectId": "$objectId",
"objectName": "$objectName",
"objectAltName": "$objectAltName",
"objectDescription": "$objectDescription",
"pluginName": "$pluginName",
"pluginDescription": "$pluginDescription",
"policyId": "$policyId",
"policyName": "$policyName",
"thresholdId": "$thresholdId",
"thresholdName": "$thresholdName"
}
```

a. Click on Test Webhook button to send a test HTTP event to the probe. You should test both Triggering condition and Clear condition webhooks and make sure the response code received is `200`.

   Note: If a response code of `400 Bad request` is received, check the JSON template set in the Body section and make sure that there is no JSON syntax error. If the status code is `0`, there could be a problem with the HTTPS connection.

Deploying the Probe for Turbonomic Integration

IBM Netcool/OMNIbus Probe for IBM Probe for Turbonomic Integration creates the webhook endpoints to receive notification in the form of HTTP POST requests from IBM Turbonomic Integration.

Note: Since the release of IBM Cloud Pak AIOps 4.1, you can now use the Generic Webhook Connector to integrate IBM Cloud Pak for AIOps with IBM Turbonomic to digest events. Using the Webhook Connector is preferred over installing a probe to connect with IBM Turbonomic as it simplifies the setup process. For details seeCreating Generic Webhook connections

Note: The IBM Netcool Operations Insight Event Integrations Operator must be deployed before deploying this integration. See "Installing the IBM Netcool Operations Insight Event Integrations Operator" on page 21.

To deploy a new instance of the Generic Webhook Probe, review the custom resource YAML template supplied with the operator package, update the parameters if required, and save the configurable parameters in a file, for example `probes_v1_webhookprobe_turbonomic_integration.yaml`.

Then deploy the custom resource using the updated file:

```
kubectl apply -f deploy/crs/probes_v1_webhookprobe_turbonomic_integration.yaml --namespace
<NAMESPACE>
```

Note: If you are installing using the CLI, you can get the custom resource Yaml file from the `ibm-netcool-integrations/inventory/netcoolIntegrationsOperatorSetup/files/op-cli/deploy/crs` directory in our CASE archive. If you are installing using the Operator Lifecycle Manager (OLM) Web console, the custom resource is provided in the YAML view.

Prerequisites

The following systems is a prerequisite for this integration.

1. IBM Cloud Pak for AIOps 3.5 or IBM Netcool Operations Insight (NOI) 1.6.6 in Openshift Container Platform.

   Note: The Probe for IBM Turbonomic integration is provided as a Technology Preview for IBM Netcool Operations Insight (NOI) 1.6.5.

   However, the integration is available for production and support under the IBM Cloud Pak for AIOps 3.4 offering. See the following:

   - Announcement Letter of IBM Cloud Pak for AIOps 3.5
   - Announcement Letter of IBM Netcool Operations Insight V1.6
2. IBM Turbonomic Application Resource Management.

   The integration has been certified with Turbonomic 8.6.3.

Known limitations for the Turbonomic integration

- Turbonomic webhook workflow can only be configured using the API.
- Turbonomic policy can be configured with one workflow per stage.
- Turbonomic sends events to one workflow if there are multiple similar policies which generated the action. If there is already a policy configured to notify another orchestrator, the probe may not be receiving an event.
- When a Turbonomic action is created, an event will be created. When the action is executed in Turbonomic, the webhook will also be notified and the event is cleared the NOI Alert Viewer. However an action can be closed in Turbonomic without being executed (for example, an erroneous condition which is no longer occurring) in which case the probe will not receive any notification from Turbonomic. This in effect will leave the event open indefinitely in NOI Alert Viewer and it will need to be cleared manually.

Pre-installation tasks

The Probe for SevOne, Probe for Turbonomics, and Probe for Datadog have the same pre-installation requirements. See "Common pre-installation tasks for the IBM SevOne, IBM Turbonomic, and Datadog probes" on page 139 before proceeding with the following sections.

Configuring and installing the Probe for Turbonomic integration

Follow the instructions below to create the required secrets, configure the `WebhookProbe` custom resource and install the probe in the namespace.

1. Create a secret with the ObjectServer credentials for the probe to authenticate with the ObjectServer.

   ```
   PROBE_OMNI_SECRET=noi-probe-secret
   kubectl create secret generic $PROBE_OMNI_SECRET --from-
   literal=AuthUserName=$NOI_OMNI_USERNAME --from-literal=AuthPassword=$NOI_OMNI_PASSWORD --
   from-file=tls.crt=tls.crt
   ```
2. Create a secret for the probe to use for basic authentication.

   ```
   PROBE_AUTH_SECRET=turbonomic-probe-client-basic-auth
   kubectl create secret generic
   ```

```
$PROBE_AUTH_SECRET --from-literal=serverBasicAuthenticationUsername=<username>  --from-
literal=serverBasicAuthenticationPassword=<password>
```

Where `<username>` is a user name and `<password>` is the password for Turbonomic to use as basic authentication.

3. Create a Network Policy in the NOI namespace to allow the probe to access the TLS Proxy service and port in NOI namespace. **Note**: Review any other Network Policy that may be denying access to the ObjectServer TLS Proxy pod and update the policy to allow ingress connection to the TLS Proxy pod.

```
cat << EOF | tee >(kubectl apply -f -) | cat
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: noi-allow-proxy
  namespace: ${NOI_NAMESPACE}
spec:
  podSelector:
    matchLabels:
      app.kubernetes.io/instance: ${NOI_INSTANCE}
      app.kubernetes.io/name: proxy
  ingress:
    - ports:
        - protocol: TCP
          port: ${NOI_PROXY_PRIMARY_PORT}
        - protocol: TCP
          port: ${NOI_PROXY_BACKUP_PORT}
  policyTypes:
    - Ingress
EOF
```

4. Create a Probe for Turbonomic Integrations with the `WebhookProbe` custom resource by running the commands below.

```
PROBE_TURBONOMIC_INSTANCE=turbonomic-probe

cat << EOF | tee >(kubectl apply -f -) | cat
apiVersion: probes.integrations.noi.ibm.com/v1
kind: WebhookProbe
metadata:
  name: ${PROBE_TURBONOMIC_INSTANCE}
  labels:
    app.kubernetes.io/name: ${PROBE_TURBONOMIC_INSTANCE}
    app.kubernetes.io/managed-by: netcool-integrations-operator
    app.kubernetes.io/instance: ${PROBE_TURBONOMIC_INSTANCE}
  namespace: ${NAMESPACE}
spec:
  helmValues:
    netcool:
      backupHost: '${NOI_PROXY_SVC}.${NOI_NAMESPACE}.svc'
      backupPort: ${NOI_PROXY_BACKUP_PORT}
      backupServer: 'AGGB'
      connectionMode: SSLAndAuth
      primaryHost: '${NOI_PROXY_SVC}.${NOI_NAMESPACE}.svc'
      primaryPort: ${NOI_PROXY_PRIMARY_PORT}
      primaryServer: 'AGGP'
      secretName: '${PROBE_OMNI_SECRET}'
    probe:
      jsonParserConfig:
        notification:
          jsonNestedHeader: ''
          jsonNestedPayload: ''
          messageDepth: 3
          messageHeader: ''
          messagePayload: json
      integration: turbonomic
      enableTransportDebugLog: false
      messageLevel: debug
    ingress:
      enabled: true
      host: ''
    arch: amd64
    webhook:
      uri: /probe/turbonomic
      serverBasicAuthenticationCredentialsSecretName: '${PROBE_AUTH_SECRET}'
      tls:
        enabled: true
        secretName: ''
  license:
    accept: true
  version: 3.1.0
EOF
```

5. Verify that the probe pod is running.

```
kubectl get pods -l app.kubernetes.io/instance=$PROBE_TURBONOMIC_INSTANCE
```

Installing the Gateway

To integrate with IBM Cloud Pak for AIOps, follow the instructions in Installing IBM Cloud Pak for AIOps to setup event flow into IBM Cloud Pak for AIOps. You may skip this step if there is already a gateway instance installed.

Obtaining the Probe Webhook URL

Use the following command to get the probe webhook URL which will be used to configure a Webhook destination in IBM Turbonomic.

```
PROBE_HOSTNAME=$(oc get route $PROBE_TURBONOMIC_INSTANCE-mb-webhook -o jsonpath='{.spec.host}')
PROBE_URI=$(oc get route $PROBE_TURBONOMIC_INSTANCE-mb-webhook -o jsonpath='{.spec.path}')
PROBE_WEBHOOK_URL=https://$PROBE_HOSTNAME$PROBE_URI
echo "$PROBE_WEBHOOK_URL"
```

Configuring IBM Turbonomic to forward events to the Probe for Turbonomic

IBM Turbonomic must be configured with a new webhook configuration to send alerts to the Probe for Turbonomic.

Creating a workflow

1. Set the following variables.

```
    TURBONOMIC_HOSTNAME=<api endpoint hostname>
    TURBONOMIC_ADMIN_USER=<turbonomic username>
    TURBONOMIC_ADMIN_PASSWORD=<turbonomic password>
    BASIC_AUTH_USERNAME=<probe basic authentication username>
    BASIC_AUTH_PASSWORD=<probe basic authentication password>
```

2. Get the JSESSIONID token.

```
    JSESSIONID=$(curl \
    --silent \
    --cookie-jar - \
    --insecure \
    https://${TURBONOMIC_HOSTNAME}/api/v3/login?hateoas=true \
    --data "username=${TURBONOMIC_ADMIN_USER}&password=${TURBONOMIC_ADMIN_PASSWORD}" \
    | awk '/JSESSIONID/{print $7}')
```

3. Create a workflow.

```
    curl \
    "https://${TURBONOMIC_HOSTNAME}/api/v3/workflows" \
    --insecure \
    --compressed \
    --header 'Accept: application/json' \
    --header 'Content-Type: application/json' \
    --header "cookie: JSESSIONID=${JSESSIONID}" \
    --request POST \
    -vv \
    --data @- <<EOF
    {
    "displayName": "Event Manager Probe",
    "className": "Workflow",
    "description": "Event Manager Webhook Probe",
    "discoveredBy": {
         "readonly": false
     },
    "type": "WEBHOOK",
    "typeSpecificDetails": {
      "url": "${PROBE_WEBHOOK_URL}",
      "method": "POST",
      "authenticationMethod": "BASIC",
      "username": "${BASIC_AUTH_USERNAME}",
      "password": "${BASIC_AUTH_PASSWORD}",
      "template": "\$converter.toJson(\$action)",
      "trustSelfSignedCertificates": true,
      "headers": [
         {
           "name": "Content-type",
           "value": "application/json"
         }
       ],
      "type": "WebhookApiDTO"
      }
    }
    EOF
```

4. Get the workflow UID for later use.

```
    WORKFLOW_UUID=$(curl \
    "https://${TURBONOMIC_HOSTNAME}/api/v3/workflows" \
    --insecure \
    --compressed \
    --header 'Accept: application/json' \
    --header 'Content-Type: application/json' \
    --header "cookie: JSESSIONID=${JSESSIONID}" \
    --request GET | jq -c ".[] | select(.typeSpecificDetails.url | contains(\"$
{PROBE_WEBHOOK_URL}\")) | .uuid" | tr -d '"')
```

Creating a policy

1. Login to Turbonomic console.
2. Navigate to the **Settings** page and then choose **Policies**.
3. Click **New Automation Policy** and then select the policy type. Choose **Virtual Machine**.

   This sets the type of entity that your policy will affect. For more information see Working with Policies .

   - Name the policy.
   - **Add Scope**: The scope determines which entities this policy will affect. Choose one or more groups, or create new groups and add them to the policy scope. These groups match the type of entity you have set for the policy.
   - **Policy Schedule**: The **Select Schedule** fly-out lists all the schedules that are currently defined for your instance of Turbonomic.

     Expand a schedule entry to see its details. You can skip this step if you do not need the schedule.
   - **Automation and Orchestration**: You can define automation and orchestration settings for different action types within the same policy.
     - **Action Type**: View the list of actions that are viable for the policy and make your selections.
     - **Action Generation and Acceptance**:

       Do not Generate Turbonomic: Never considers your selected actions in its calculations. For example, if you do not want to generate resize actions for VMs in the policy, analysis will still drive toward the desired state, but will do so without considering resizing.

       Generate Turbonomic: Generates your selected actions to address or prevent problems.

       Choose from the following Action Acceptance modes to indicate how you would like the actions to execute:

       Recommend: Recommend the action so a user can execute it via the given hypervisor or by other means

       Manual: Recommend the action, and provide the option to execute that action through the Turbonomic user interface

Automatic: Execute the action automatically For automated resize or move actions on the same entity, Turbonomic waits five minutes between each action to avoid failures associated with trying to execute all actions at once. Any action awaiting execution stays in queue.

- **Before Execution, Action Execution and After Execution**: By default, generated actions execute without the need for orchestration. Turbonomic gives you the ability to set up orchestration to affect the execution of actions.

  The example below shows how to apply **Action Execution**, which would send close event action through workflow.

  – **After Execution**: The default is `Do Nothing`.

    Other options include:

    Create Record in Orchestration: Turbonomic registers the action in the Orchestration log, showing that the given action has been executed. For example: A final ActionID could be created and with the final ActionState, the Type and Severity would be updated accordingly in waiops. Those actions would also be updated to Waiops, and it would handle with the alerts with updated state: closed or cleared or keep open.

    Use Action Script Run: An action script that is set up for the POST entrypoint. The action script name must match the entity type and action type.

4. Save the policy.
5. Action events will be sent to the probe when an action is executed.

Triggering a test event

1. Get a list of Actions by market to test out the workflow. Select an action from the list and copy the UUID which will be used to trigger an action event manually using API.

```
    curl \
    "https://${TURBONOMIC_HOSTNAME}/api/v3/markets/Market/actions?ascending=true" \
    --insecure \
    --compressed \
    --header 'Accept: application/json' \
    --header 'Content-Type: application/json' \
    --header "cookie: JSESSIONID=${JSESSIONID}" \
    --request GET
```

2. To test triggerring an action event based on the UUID in the workflow, use one of the action UUID from the previous command in the actionId attribute value in the command below:

```
    curl --location --request POST "https://${TURBONOMIC_HOSTNAME}/api/v3/workflows/$
{WORKFLOW_UUID}" \
    --header 'Content-Type: application/json' \
    --header "Cookie: JSESSIONID=$JSESSIONID" \
    --insecure \
    --data-raw '{
        "operation": "TEST",
        "actionId": "637553640602793"
    }'
```

Deploying the Probe for Datadog Integration

IBM Netcool/OMNIbus Probe for Datadog Integration sends HTTP GET requests to Datadog to retrieve events based on the time frame set by **`start`** and **end** mandatory query parameters in the URI.

Note: The IBM Netcool Operations Insight Event Integrations Operator must be deployed before deploying this integration. See “Installing the IBM Netcool Operations Insight Event Integrations Operator” on page 21.

To deploy a new instance of the Generic Webhook Probe, review the custom resource YAML template supplied with the operator package, update the parameters if required, and save the configurable parameters in a file, for example `probes_v1_webhookprobe_datadog_integration.yaml`.

Then deploy the custom resource using the updated file:

```
kubectl apply -f deploy/crs/probes_v1_webhookprobe_datadog_integration.yaml --namespace
<NAMESPACE>
```

Note: If you are installing using the CLI, you can get the custom resource Yaml file from the `ibm-netcool-integrations/inventory/netcoolIntegrationsOperatorSetup/files/op-cli/deploy/crs` directory in our CASE archive. If you are installing using the Operator Lifecycle Manager (OLM) Web console, the custom resource is provided in the YAML view.

Prerequisites

The following systems are prerequisites for this integration.

1. IBM Cloud Pak for AIOps 4.1.0 or IBM Netcool Operations Insight (NOI) 1.6.9 in Openshift Container Platform.
2. Datadog: Cloud Monitoring as a Service

Pre-installation tasks

The Probe for SevOne, Probe for Turbonomics, and Probe for Datadog have the same pre-installation requirements. See "Common pre-installation tasks for the IBM SevOne, IBM Turbonomic, and Datadog probes" on page 139 before proceeding with the following sections.

Configuring and installing the Probe for Datadog integration

Follow the instructions below to create the required secrets, configure the `WebhookProbe` custom resource and install the probe in the namespace.

1. Create a secret with the ObjectServer credentials for the probe to authenticate with the ObjectServer.

```
PROBE_OMNI_SECRET=noi-probe-secret
kubectl create secret generic $PROBE_OMNI_SECRET --from-
literal=AuthUserName=$NOI_OMNI_USERNAME --from-literal=AuthPassword=$NOI_OMNI_PASSWORD --
from-file=tls.crt=tls.crt
```

2. Create a secret for the Datadog certificate.

```
    echo -n | openssl s_client -connect api.datadoghq.com:443 | sed -ne '/-BEGIN
CERTIFICATE-/,/-END CERTIFICATE-/p'  > datadog.pem
    PROBE_CLIENT_TLS_SECRET=datadog-probe-client-tls-cert
    kubectl create secret generic $PROBE_CLIENT_TLS_SECRET --from-file=server.crt=datadog.pem
```

3. Create a secret for the transport properties.

```
    TRANSPORT_PROPS=restMultiChannelHttpTransport.json
    cat << EOF | tee > $TRANSPORT_PROPS
    {
      "GLOBAL":
      {
        "httpVersion":"1.1",
        "httpHeaders":"",
        "responseTimeout":"60",
        "securityProtocol":"TLSv1.2",
        "keepTokens":"",
        "autoReconnect":"OFF",
        "gatherSubsTopicInfo":"false"
      },
      "RESYNC":
      {
          "RESYNC_CHILD_BY_FDN":
          {
              "uri":"https://api.datadoghq.com:443/api/v1/events?start=+
+date_happened:<initial-start-epoch-time>++&end=++now++&page=0",
              "method":"GET",
              "headers":"Accept=application/json,DD-API-KEY=<datadog-api-key>,DD-APPLICATION-
KEY=<datadog-application-key>",
              "content":"{}",
              "contentFile" : "",
              "interval":"60",
              "attempts":"0",
                "requireSSL":"true",
                "asEventStream" : "false",
                "enablePagination" : "true"
            }
        }
    }
    EOF

    PROBE_HTTP_REQUEST=http-requests
    kubectl create secret generic $PROBE_HTTP_REQUEST --from-
file=httpRequests=$TRANSPORT_PROPS
```

Where:

`++date_happened:<initial-start-epoch-time>++` allows the probe to assign the value of `RecordData.date_happened` in the probe data backup file to the `start` query parameter to continue from previous resynchronization if `RecordData.date_happened` is available. Otherwise, the probe assigns `<initial-start-epoch-time>` to the `start` query parameter.

`<datadog-api-key>` is the Datadog API key.

`<datadog-application-key>` is Datadog Application key.

Note: The end query parameter can be set to an epoch time or ++now++ which represents the current epoch time.

4. Create a Network Policy in the NOI namespace to allow the probe to access the TLS Proxy service and port in NOI namespace. **Note**: Review any other Network Policy that may be denying access to the ObjectServer TLS Proxy pod and update the policy to allow ingress connection to the TLS Proxy pod.

```
cat << EOF | tee >(kubectl apply -f -) | cat
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: noi-allow-proxy
  namespace: ${NOI_NAMESPACE}
spec:
  podSelector:
    matchLabels:
      app.kubernetes.io/instance: ${NOI_INSTANCE}
      app.kubernetes.io/name: proxy
  ingress:
    - ports:
        - protocol: TCP
          port: ${NOI_PROXY_PRIMARY_PORT}
        - protocol: TCP
          port: ${NOI_PROXY_BACKUP_PORT}
  policyTypes:
    - Ingress
EOF
```

5. Create a Probe for Datadog Integrations with the `WebhookProbe` custom resource by running the following commands.

```
     PROBE_DATADOG_INSTANCE=datadog-probe

     cat << EOF | tee >(kubectl apply -f -) | cat
     kind: WebhookProbe
     apiVersion: probes.integrations.noi.ibm.com/v1
     metadata:
       name: webhookprobe
       labels:
         app.kubernetes.io/name: ${PROBE_DATADOG_INSTANCE}
         app.kubernetes.io/managed-by: netcool-integrations-operator
         app.kubernetes.io/instance: ${PROBE_DATADOG_INSTANCE}
       namespace: ${NAMESPACE}
     spec:
       license:
         accept: true
       version: 3.3.0
       helmValues:
         global:
           image:
             secretName: ''
           serviceAccountName: ''
           enableNetworkPolicy: true
```

```
          image:
            useDefaultOperandImages: true
            probe: >-
              cp.icr.io/cp/noi-int/netcool-probe-
messagebus@sha256:745a7dae41f1a0c8f1517a9dc2368d227e39be4798b74175f903bc5ac940b95b
            utility: >-
              cp.icr.io/cp/noi-int/netcool-integration-
util@sha256:6e0b9883a34810c468b5386b4b6d9ed216f68df015f6622b83656d400d01dcd2
            pullPolicy: IfNotPresent
          netcool:
            connectionMode: SSLAndAuth
            primaryServer: 'AGGP'
            primaryHost: '${NOI_PROXY_SVC}.${NOI_NAMESPACE}.svc'
            primaryPort: 4100
            backupServer: 'AGGB'
            backupHost: '${NOI_PROXY_SVC}.${NOI_NAMESPACE}.svc'
            backupPort: 4100
            secretName: '${PROBE_OMNI_SECRET}'
          probe:
            integration: datadog
            messageLevel: warn
            setUIDandGID: false
            sslServerCommonName: ''
            locale: en_US.utf8
            enableTransportDebugLog: false
            heartbeatInterval: 10
            recordData: 'date_happened'
            configPVC:
              name: ''
              rulesFile: ''
              parserConfigFile: ''
              dataBackupFile: ''
            rulesConfigmap: ''
            jsonParserConfig:
              notification:
                messagePayload: json
                messageHeader: ''
                jsonNestedPayload: ''
                jsonNestedHeader: ''
                messageDepth: 3
              resync:
                messagePayload: 'json.events'
                messageHeader: ''
                jsonNestedPayload: ''
                jsonNestedHeader: ''
                messageDepth: 3
            httpClientCredentialsSecretName: ''
            initialResync: true
            resyncInterval: 120
            poddisruptionbudget:
              enabled: false
              minAvailable: 1
            selfMonitoring:
              discardHeartbeatEvent: false
              populateMasterProbeStat: false
              logProbeStat: false
              generateThresholdEvents: false
            configs: 'ProbeWatchHeartbeatInterval:0 EventLoadProfiling:''true'''
          ingress:
            enabled: false
            host: ''
          webhook:
            uri: /probe
            httpVersion: '1.1'
            respondWithContent: 'OFF'
            validateBodySyntax: 'ON'
            validateRequestURI: 'ON'
            idleTimeout: 180
            tls:
              enabled: true
              secretName: ''
            serverBasicAuthenticationCredentialsSecretName: ''
            replicaCount: 1
            autoscaling:
              enabled: false
              minReplicas: 1
              maxReplicas: 3
              cpuUtil: 60
          httpClient:
            enabled: true
            host: ''
            port: 80
            sslSecretName: '$PROBE_CLIENT_TLS_SECRET'
            httpRequestsSecretName: '$PROBE_HTTP_REQUEST'
          resources:
            requests:
              cpu: 100m
              memory: 128Mi
              ephemeral-storage: 100Mi
            limits:
              cpu: 200m
              memory: 256Mi
              ephemeral-storage: 200Mi
          arch: amd64
     EOF
```

6. Verify that the probe pod is running.

```
kubectl get pods -l app.kubernetes.io/instance=$PROBE_DATADOG_INSTANCE
```

7. If you want to store the probe data backup file in Persistent Volume Claim (PVC), follow the instructions in https://www.ibm.com/docs/en/netcoolomnibus/8?topic=noieio-providing-custom-message-bus-probe-configuration-files-in-persistent-volume. Then set `configPVC.name` to a pre-created PVC name and `configPVC.dataBackupFile` to the data backup filename.

Known limitations for the Datadog integration

Probe only supports the https://api.datadoghq.com/api/v1/events v1 API to get events from Datadog.

Appendix A. Notices and Trademarks

This appendix contains the following sections:

- Notices
- Trademarks

Notices

Trademarks

SC28-3144-17