IBM Tivoli Netcool/OMNIbus Message Bus Probe for Kafka Integrations Helm Chart 2.1.0

Reference Guide
October 31, 2019

SC27-9524-02

Note

Before using this information and the product it supports, read the information in Appendix A, “Notices and Trademarks,” on page 35.

Edition notice

This edition (SC27-9524-02) applies to version 2.1.0 of IBM Tivoli Netcool/OMNIbus Message Bus Probe for Kafka Integrations Helm Chart and to all subsequent releases and modifications until otherwise indicated in new editions.

This edition replaces SC27-9524-01.

Contents

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 Message Bus Probe for Kafka Integrations Helm Chart 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/omnibus/helms/common/Helms.html

Table 1. Document modification history

Document version	Publication date	Comments
SC27-9524-00	October 11, 2018	First IBM publication.
SC27-9524-01	February 28, 2019	Guide updated for version 2.0.0 of the helm chart. Helm chart now supports ICP 3.1.x. The following topics were updated: • “Obtaining the PPA package” on page 1 • “Prerequisites” on page 2 • “Resources required” on page 2 • “Uninstalling the chart” on page 11 • “Configurable parameters” on page 12 • “Limitations” on page 33 The following topic was added: • “PodSecurityPolicy requirements” on page 3

Table 1. Document modification history (continued)		
Document version	**Publication date**	**Comments**
SC27-9524-02	October 31, 2019	Guide updated for version 2.1.0 of the helm chart. The following topics were updated: • "Obtaining the PPA package" on page 1 • "Chart details" on page 1 • "Prerequisites" on page 2 • "Configurable parameters" on page 12 • "Configuring the Message Bus Kafka probe to connect in secured mode" on page 27 • "Uninstalling the chart" on page 11 • "Limitations" on page 33 The following topics were added: • "Secure probe and ObjectServer communication requirements" on page 6 • "Integrating the Message Bus Kafka probe with Apache Kafka" on page 17 • "Resources required" on page 2 • "Integrating the Message Bus Kafka probe with Microsoft Azure Event Hubs for Kafka" on page 21 • "Integrating the Message Bus Kafka probe with IBM Event Streams" on page 24

Chapter 1. Message Bus Probe for Kafka Integrations Helm Chart

The Message Bus Probe for Kafka Integrations Helm Chart allows you to deploy the IBM Netcool/OMNIbus Message Bus Probe for Kafka onto Kubernetes. This probe processes Kafka events from a Kafka server to a Netcool Operations Insight operational dashboard.

This guide contains the following sections:

The Knowledge Center contains the following additional topics that contain information that is common to all Helm Charts:

- Specifying the image repository
- Loading PPA packages to IBM Cloud Private
- Exposing the probe service
- Upgrading to a new version of the probe helm charts
- Changing the service type during a helm upgrade

Obtaining the PPA package

You can download the installation package from the **IBM Passport Advantage** website.

Use the **Find by part number** field to search for the following part number: CC43PEN

Chart details

This helm chart deploys the IBM Netcool/OMNIbus Message Bus Probe for Kafka onto Kubernetes. This probe can receive messages from a Kafka server.

This chart has been validated to run on the following platforms:

- IBM Cloud Private 3.2.0
- IBM Cloud Private 3.1.2
- IBM Cloud Private 3.2.0 on Red Hat OpenShift Container Platform 3.11

- IBM Cloud Private 3.1.2 on Red Hat OpenShift Container Platform 3.10

This chart can be deployed more than once on the same namespace. The chart has been verified against Apache Kafka 2.11-2 (Scala 2.11)

This chart has been validated to run against the following Kafka services:

- IBM Event Streams 2019.2.1
- Microsoft Azure Event Hubs for Kafka

Prerequisites

This solution requires the following applications:

- IBM Tivoli Netcool/OMNIbus ObjectServer to be created and running prior to installing the probe either on IBM Cloud Private (ICP) or on-premise.
  - For ICP, IBM Netcool Operations Insight 1.6.0.1 is required. See the following topic on the IBM Knowledge Center: Installing on IBM Cloud Private.
  - For on-premise, IBM Tivoli Netcool/OMNIbus 8.1 is required. See the following topic on the IBM Knowledge Center: Creating and running ObjectServers.
- Scope-based Event Grouping automation to be installed. The probe requires several table fields to be installed in the ObjectServer. For on-premise installation, see installation instructions on the IBM Knowledge Center: IBM Knowledge Center - 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.
- Kubernetes 1.10.

Note :

The chart must be installed by a Administrator to perform the following tasks:

- Enable Pod Disruption Budget policy.
- Retrieve and edit sensitive information from a secret such as the credentials to use to authenticate with the Object Server or replace the Key database files for secure communications with the Object Server.

The chart must be installed by a Cluster Administrator to perform the following task in addition to the one above:

- Obtain the Node IP using `kubectl get nodes` command if using the NodePort service type.
- Create a new namespace with custom PodSecurityPolicy if necessary. For details see "PodSecurityPolicy requirements" on page 3.

A custom service account must be created in the namespace for this chart. Perform one of the following actions:

- Have the Cluster Administrator pre-create the custom service account in the namespace. This installation requires the service account name to specified to install the chart and can be done by an Administrator.
- Install the chart with the service account being automatically created. This installation must be performed by the Cluster Administrator.

To connect the probe to a Kafka server using a secure connection, the Kubernetes Secrets for the client Truststore certificate and the client Keystore certificate must be created before deploying the helm chart. Creating the Secrets requires the Operator role or higher. See "Configuring the Message Bus Kafka probe to connect in secured mode" on page 27.

Resources required

This solution requires the following resources:

- CPU Requested : 250m (250 millicpu)
- Memory Requested : 256Mi (~ 268 MB)

PodSecurityPolicy requirements

This chart requires a PodSecurityPolicy to be bound to the target namespace prior to installation. You can choose either a predefined PodSecurityPolicy or have your cluster administrator create a custom PodSecurityPolicy for you.

The predefined PodSecurityPolicy name `ibm-restricted-psp` has been verified for this chart, see IBM Cloud Pak Pod Security Policy Definitions. If your target namespace is bound to this PodSecurityPolicy, you can proceed to install the chart. The predefined PodSecurityPolicy definitions can be viewed here: https://github.com/IBM/cloud-pak/blob/master/spec/security/psp/README.md

This chart also defines a custom PodSecurityPolicy which can be used to finely control the permissions/capabilities needed to deploy this chart. You can enable this custom PodSecurityPolicy using the ICP user interface or the supplied instructions/scripts in the `pak_extension pre-install` directory. For detailed steps on creating the PodSecurityPolicy see https://www.ibm.com/support/knowledgecenter/SSHTQ/omnibus/helms/all_helms/wip/reference/hlm_common_psp.html

From the user interface, you can copy and paste the following snippets to enable the custom Pod Security Policy

- From the user interface, you can copy and paste the following snippets to enable the custom PodSecurityPolicy:
  - Custom PodSecurityPolicy definition:

```
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
  annotations:
    kubernetes.io/description: "This policy is based on the most restrictive policy,
    requiring pods to run with a non-root UID, and preventing pods from accessing the host."
    seccomp.security.alpha.kubernetes.io/allowedProfileNames: docker/default
    seccomp.security.alpha.kubernetes.io/defaultProfileName: docker/default
    cloudpak.ibm.com/version: "1.1.0"
  name: ibm-netcool-probe-messagebus-kafka-prod-psp
spec:
  allowPrivilegeEscalation: false
  forbiddenSysctls:
  - '*'
  fsGroup:
    ranges:
    - max: 65535
      min: 1
    rule: MustRunAs
  requiredDropCapabilities:
  - ALL
  runAsUser:
    rule: MustRunAsNonRoot
  runAsGroup:
    rule: MustRunAs
    ranges:
    - min: 1
      max: 65535
  seLinux:
    rule: RunAsAny
  supplementalGroups:
    ranges:
    - max: 65535
      min: 1
    rule: MustRunAs
  volumes:
  - configMap
  - emptyDir
  - projected
  - secret
  - downwardAPI
  - persistentVolumeClaim
```

  - Custom ClusterRole for the custom PodSecurityPolicy:

```
    apiVersion: rbac.authorization.k8s.io/v1
    kind: ClusterRole
    metadata:
      name: ibm-netcool-probe-messagebus-kafka-prod-clusterrole
    rules:
    - apiGroups:
      - policy
      resourceNames:
      - ibm-netcool-probe-messagebus-kafka-prod-psp
      resources:
      - podsecuritypolicies
      verbs:
      - use
```

– RoleBinding for all service accounts in the current namespace. Replace {{ NAMESPACE }} in the template with the actual namespace:

```
    apiVersion: rbac.authorization.k8s.io/v1
    kind: RoleBinding
    metadata:
      name: ibm-netcool-probe-messagebus-kafka-prod-rolebinding
    roleRef:
      apiGroup: rbac.authorization.k8s.io
      kind: ClusterRole
      name: ibm-netcool-probe-messagebus-kafka-prod-clusterrole
    subjects:
    - apiGroup: rbac.authorization.k8s.io
      kind: Group
      name: system:serviceaccounts:{{ NAMESPACE }}
```

- From the command line, you can run the setup scripts included under `pak_extensions`.

  As a cluster administrator, the pre-install scripts and instructions are in the following location:

  `pre-install/clusterAdministration/createSecurityClusterPrereqs.sh`

  As team admin/operator the namespace scoped scripts and instructions are in the following location:

  `pre-install/namespaceAdministration/createSecurityNamespacePrereqs.sh`

Security context constraints

On the Red Hat OpenShift Container Platform, this chart requires a SecurityContextConstraints to be bound to the target namespace prior to installation. To meet this requirement there may be cluster scoped as well as namespace scoped pre and post actions that need to occur.

The predefined SecurityContextConstraints name: `ibm-restricted-scc` has been verified for this chart, see https://github.com/IBM/cloud-pak/blob/master/spec/security/scc/README.md. If your target namespace is bound to this SecurityContextConstraints resource you can proceed to install the chart.

This chart also defines a custom SecurityContextConstraints which can be used to finely control the permissions/capabilities needed to deploy this chart. You can enable this custom SecurityContextConstraints resource using the supplied instructions/scripts in the pak_extension pre-install directory.

From the user interface, you can copy and paste the following snippets to enable the custom `SecurityContextConstraints`:

- – Custom `SecurityContextConstraints` definition:

```
apiVersion: security.openshift.io/v1
kind: SecurityContextConstraints
metadata:
  annotations:
    kubernetes.io/description: "This policy is the most restrictive,
      requiring pods to run with a non-root UID, and preventing pods from accessing the
host."
    cloudpak.ibm.com/version: "1.1.0"
  name: ibm-netcool-probe-messagebus-kafka-prod-scc
allowHostDirVolumePlugin: false
allowHostIPC: false
allowHostNetwork: false
```

```
allowHostPID: false
allowHostPorts: false
allowPrivilegedContainer: false
allowPrivilegeEscalation: false
allowedCapabilities: null
allowedFlexVolumes: null
allowedUnsafeSysctls: null
defaultAddCapabilities: null
defaultAllowPrivilegeEscalation: false
forbiddenSysctls:
  - "*"
fsGroup:
  type: MustRunAs
  ranges:
  - max: 65535
    min: 1
readOnlyRootFilesystem: false
requiredDropCapabilities:
- ALL
runAsUser:
  type: MustRunAsNonRoot
seccompProfiles:
- docker/default
# This can be customized for seLinuxOptions specific to your host machine
seLinuxContext:
  type: RunAsAny
# seLinuxOptions:
#   level:
#   user:
#   role:
#   type:
supplementalGroups:
  type: MustRunAs
  ranges:
  - max: 65535
    min: 1
# This can be customized to host specifics
volumes:
- configMap
- downwardAPI
- emptyDir
- persistentVolumeClaim
- projected
- secret
# If you want a priority on your SCC -- set for a value more than 0
# priority:
```

- From the command line, you can run the setup scripts included under `pak_extensions`.

  As a cluster administrator, the pre-install scripts and instructions are in the following location:

  `pre-install/clusterAdministration/createSecurityClusterPrereqs.sh`

  As team admin/operator the namespace scoped scripts and instructions are in the following location:

  `pre-install/namespaceAdministration/createSecurityNamespacePrereqs.sh`

Creating prerequisite resources for SecurityContextConstraints using the command line

1. Login using `cloudctl` followed by `oc`. In the example below, `MasterNode_IP` refers to the IP address of the master node and `ICP_port` refers to the ICP port.

   ```
   cloudctl login -a https://<MasterNode_IP>:<ICP_port>
   oc login
   ```

2. Extract the pre-installation scripts from the archive under the ibm_cloud_pak/pak_extension directory. Example below shows how to extract the scripts from *ibm-noi-probe-3.10.4.1-x86.tgz* archive.

   ```
   tar xvf ibm-noi-probe-3.10.4.1-x86.tgz ibm_cloud_pak
   ```

3. As a cluster administrator, run the `createSecurityClusterPrereqs.sh` script and provide the target namespace as an argument. This script creates the SecurityContextConstraints and ClusterRole

resources. The example command below runs the script on a namespace called `my-probe-namespace`.

```
cd ibm_cloud_pak/pak_extensions
./pre-install/clusterAdministration/createSecurityClusterPrereqs.sh my-probe-namespace
```

4. As an administrator or cluster administrator, run the namespace scoped `createSecurityNamespacePrereqs.sh` script and provide the target namespace as an argument. This script creates the RoleBinding resource for service accounts in the target namespace. The namespace must be created prior to running this script. The example command below runs the script on a namespace called `my-probe-namespace`.

```
cd ibm_cloud_pak/pak_extensions
./pre-install/namespaceAdministration/createSecurityNamespacePrereqs.sh my-probe-namespace
```

5. Verify that the custom SCC has been created.

```
oc get scc | grep <SCC name>
```

6. You can now proceed to install the chart in the namespace with the custom SecurityContextConstraints.

Secure probe and ObjectServer communication requirements

There are several mechanisms to secure Netcool/OMNIbus. 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 is dependent 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, refer to Running the ObjectServer in secure mode on the IBM Knowledge Center. For more details on SSL protected communications, refer to Using SSL for client and server communications on the IBM Knowledge Center.

The chart must be configured according to the server components setup in order to establish a secured connection with or without SSL. This can be configured by setting the `messagebus.netcool.connectionMode` chart parameter with one of the following 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.

To secure the communications between probe clients and the ObjectServer, you must perform the following tasks that must be completed before installing the chart:

1. “Determine files required for the secret” on page 7
2. “Preparing credential files for authentication” on page 7
3. “Preparing the key database file for SSL communication” on page 8
4. “Create the probe-server communication secret” on page 10

If you are using the `Default` mode, you can skip these steps and proceed configuring the chart with your ObjectServer connection details.

Note : There are several known limitations listed in the Limitations section when securing probe communications.

Determine files required for the secret

For `AuthOnly`, `SSLOnly` and `SSLAndAuth` options, a secret is required to support the secured connection prior installing the chart. The secret name should be set in the `messagebus.netcool.secretName` parameter when configuring the chart 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 chart.

For secured mode (`AuthOnly`), the following fields are required:

- `encryption.keyfile` - An encryption key file to decrypt your password.
- `AuthUserName` - Specify the username to authenticate with the ObjectServer.
- `AuthPassword` - Specify the password for the specified user. It is recommended to encrypt this string using the encryption key file above.

For SSL enabled mode (`SSLOnly`), the following fields are required:

- `omni.kdb` - A Key Database file. This Key Database file should contain the root Certificate Authority certificate used by the server components.
- `omni.sth` - The Key Database Stash file. This file is required for automatic authentication.

For secured and SSL enabled mode (`SSLAndAuth`), the following fields are required:

- `encryption.keyfile` - An encryption key file to decrypt your password.
- `AuthUserName` - Specify the username to authenticate with the ObjectServer.
- `AuthPassword` - Specify the password of the specified user. It is recommended to encrypt this string using the encryption key file above.
- `omni.kdb` - A Key Database file. This Key Database file should contain the root Certificate Authority certificate used by the server components.
- `omni.sth` - The Key Database Stash file. This file is required for automatic authentication.

You only need to create a single secret with the required fields which can be used by both the Logstash and Prometheus probes.

Preparing credential files for authentication

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.

If you are connecting to an ObjectServer on IBM Cloud Private, you need to extract the CA certificate from the proxy deployed with the ibm-netcool-prod 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 event sources.

Note :

This section is only applicable if you are using either the `AuthOnly` or `SSLAndAuth` connection mode.

Some of steps below use utility commands in the probe image. Update the image repository in the each command to pull the image from IBM Cloud Private's private registry or pull the image to your local file system first. For more details about how to pull the image to your local file system, see Pushing and pulling images.

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 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 uses `/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_user" > /tmp/probe/username.txt
   ```

4. If an encryption key file is not provided by the ObjectServer administrator, you (administrator) need to create a new encryption key file to encrypt the password. Otherwise, you can skip this step. To create an encryption file using the `nco_keygen` command from the probe Docker image, use the following commands:

```
# Start the probe container and create an encryption key file in the mounted directory.
# Use valid 256 for the key length.
$ docker run -e LICENSE=accept \
--entrypoint /bin/bash -it \
-v /tmp/probe:/home/netcool netcool-probe-messagebus:10.0.5.0-amd64 \
-c "/opt/IBM/tivoli/netcool/omnibus/bin/nco_keygen \
-o /home/netcool/encryption.keyfile \
-l 256"
```

5. Encrypt the password string with the encryption key file using the `nco_aes_crypt` tool with `AES_FIPS` cipher and output the encrypted password into a file called `password.txt` in the mounted directory. Example command to encrypt a password (`AVeryStrongPassw0rd`) is shown below. Replace the password string in the command with your own.

```
# Encrypt the password
$ docker run -e LICENSE=accept \
--entrypoint /bin/bash -it \
-v /tmp/probe:/home/netcool \
netcool-probe-messagebus:10.0.5.0-amd64 \
-c "/opt/IBM/tivoli/netcool/omnibus/bin/nco_aes_crypt \
-c AES_FIPS \
-k /home/netcool/encryption.keyfile AVeryStrongPassw0rd \
-o /home/netcool/password.txt"
```

6. Verify that the following files are created in the temporary workspace directory:
   - `encryption.keyfile`
   - `password.txt`
   - `username.txt`

```
$ ls -1 /tmp/probe/
encryption.keyfile
password.txt
username.txt
```

Preparing the key database file 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 set up the key database files and add the server certificate.

Note :

This is section is only applicable if you are using either the `SSLOnly` or `SSLAndAuth` connection.

Some of steps below use utility commands in the probe image. Update the image repository in the each command to pull the image from IBM Cloud Private's private registry or pull the image to your local file system first. For more details about how to pull the image to your local file system, see Pushing and pulling images.

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, see Extracting certificates

from a key database. 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" label into the file ncoms-ca-cert.arm:

```
$NCHOME/bin/nc_gskcmd -cert -extract \
-db "$NCHOME/etc/security/keys/omni.kdb" \
-pw password \
-label "NCOMS_CA" \
-target "$NCHOME/etc/security/keys/ncoms-ca-cert.arm"
```

2. Create a temporary workspace directory if necessary. The following steps use /tmp/probe as the workspace directory.

```
$ mkdir /tmp/probe
```

3. Copy the server certificate into the workspace directory.

```
$ cp ncoms-ca-cert.arm /tmp/probe
```

4. Create a key database file with a strong password which meets the password requirements, see Creating a key database using nc_gskcmd.

   For example, to create a key database file that is valid for 366 days:

```
# Create a Key Database file
$ docker run -e LICENSE=accept \
--entrypoint /bin/bash -it \
-v /tmp/probe:/home/netcool \
netcool-probe-messagebus:10.0.5.0-amd64 \
-c "/opt/IBM/tivoli/netcool/bin/nc_gskcmd -keydb -create \
-db \"/home/netcool/omni.kdb\" \
-pw AVeryStrongKeyDBPasSw0rd \
-stash \
-expire 366"
```

5. Add the server certificate into the key database file. The following example command adds the server certificate into the ncoms-ca-cert.arm file into the key database with the label NCOMS_CA:

```
# Add server root CA into Key DB
$ docker run -e LICENSE=accept \
--entrypoint /bin/bash -it \
-v /tmp/probe:/home/netcool \
netcool-probe-messagebus:10.0.5.0-amd64 \
-c "/opt/IBM/tivoli/netcool/bin/nc_gskcmd -cert -add \
-db \"/home/netcool/omni.kdb\" \
-pw AVeryStrongKeyDBPasSw0rd \
-label \"NCOMS_CA\" \
-file \"/home/netcool/ncoms-ca-cert.arm\""
```

6. Optionally, you can verify that the server certificate has been added into the key database using the following command:

```
# Add server root CA into Key DB
$ docker run -e LICENSE=accept \
--entrypoint /bin/bash -it \
-v /tmp/probe:/home/netcool \
netcool-probe-messagebus:10.0.5.0-amd64 \
-c "/opt/IBM/tivoli/netcool/bin/nc_gskcmd -cert -list \
-db \"/home/netcool/omni.kdb\" \
-pw AVeryStrongKeyDBPasSw0rd"
Certificates found
```

```
default, - personal, ! trusted, # secret key
- NCOMS_CA
```

Create the probe-server communication secret

After the preparing the required files, use the `kubectl 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 `my-probe` as the release name to create a secret called `my-probe-comms` in the `icp-noi` namespace.

- To create a secret for secured communication (`AuthOnly` connection mode), use:

```
kubectl create secret generic my-probe-comms \
--namespace icp-noi \
--from-file=encryption.keyfile=/tmp/probe/encryption.keyfile \
--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), use:

```
kubectl create secret generic my-probe-comms \
--namespace icp-noi \
--from-file=omni.kdb=/tmp/probe/omni.kdb \
--from-file=omni.sth=/tmp/probe/omni.sth
```

- To create a secret for SSL enabled and secured communication (`SSLAndAuth` connection mode), use:

```
kubectl create secret generic my-probe-comms \
--namespace icp-noi \
--from-file=encryption.keyfile=/tmp/probe/encryption.keyfile \
--from-file=AuthUserName=/tmp/probe/username.txt \
--from-file=AuthPassword=/tmp/probe/password.txt \
--from-file=omni.kdb=/tmp/probe/omni.kdb \
--from-file=omni.sth=/tmp/probe/omni.sth
```

Note : Each field key must follow the expected names `AuthUserName`, `AuthPassword`, `encryption.keyfile`, `omni.kdb` and `omni.sth` and is case sensitive. If any of the field key is misspelled, the Pod might fail to mount to the secret to obtain the secret.

Tip : If you have a different encryption key file name, you can map the name of the file to `encryption.keyfile` when creating the secret as shown below. The sample command below renames `ProbeEncryption.keyfile` to the expected field name.

```
kubectl create secret generic my-probe-comms \
--namespace icp-noi \
--from-file=encryption.keyfile=/tmp/probe/ProbeEncryption.keyfile  \
--from-file=AuthUserName=/tmp/probe/username.txt \
--from-file=AuthPassword=/tmp/probe/password.txt \
--from-file=omni.kdb=/tmp/probe/omni.kdb \
--from-file=omni.sth=/tmp/probe/omni.sth
```

Remember to clean up this directory after you have successfully created the Kubernetes secret.

Installing the chart

There are two ways to install a Helm Release into IBM Cloud Private or Kubernetes:

- Using IBM Cloud Private Management Console UI: Provides a quick way to install a helm chart.
- Using Command Line. Allows you to perform repetitive installs using a customized `values.yaml` easily.

Using the IBM Cloud Private Management Console UI

1. Navigate to the Helm Charts Catalog. The Catalog can be accessed on the top right of the Menu.
2. Search the Catalog for the Helm Chart under the name kafka.
3. Click on the Helm Chart and then on Configure.
4. Enter all required properties and click Install to install the helm chart.

Using the command line

To install a chart with the release name *my-probe*, use the following steps:

1. Extract the helm chart archive and customize the `values.yaml` file. The configuration section lists the parameters that can be configured during installation.
2. The following command shows how to install the chart with the release name *my-probe* using the configuration specified in the `values.yaml` file. Helm searches for the `ibm-netcool-probe-messagebus-kafka-prod` chart in the helm repository called `stable`. This assumes that the chart exists in the `stable` repository.

   ```
   helm install --tls --name my-probe -f values.yaml stable/ibm-netcool-probe-messagebus-kafka-prod
   ```

   Where: *my-probe* is the release name for the chart.

   Tip : You can list all releases using `helm list --tls` or search for a chart using `helm search`.

Verifying the chart

See the instructions at the end of the helm installation for chart verification. The instructions can also be displayed by viewing the installed helm release under **Menu -> Workloads -> Helm Releases** or by running the following command:

```
helm status <release> --tls
```

Uninstalling the chart

To uninstall/delete the `my-probe` deployment:, use the following command:

```
helm delete my-probe --purge --tls
```

The command removes all the Kubernetes components associated with the chart and deletes the release.

Clean up any prerequisites that were created

As a Cluster Administrator, run the cluster administration cleanup script included under `pak_extensions` to clean up cluster scoped resources when appropriate.

```
post-delete/clusterAdministration/deleteSecurityClusterPrereqs.sh
```

As a Cluster Administrator, run the namespace administration cleanup script included under `pak_extensions` to clean up namespace scoped resources when appropriate.

```
post-delete/namespaceAdministration/deleteSecurityNamespacePrereqs.sh
```

Configuring the chart

The integration requires configuration of the chart parameters.

Configurable parameters

You use parameters to specify how the probe interacts with the device. You can override the chart's default parameter settings during installation.

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

Table 2. Configurable parameters

Parameter name	Description
messagebus.license	Use this parameter to specify the license state of the image being deployed. Enter `accept` to install and use the image. The default value is `not accepted`
messagebus.replicaCount	Use this parameter to specify the number of deployment replicas. This should be omitted when **autoscaling.enabled** is set to `true`. The default value is `1`
messagebus.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`
messagebus.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`.
messagebus.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 `netcool-probe-messagebus`
messagebus.image.tag	Use this parameter to specify the `netcool-probe-messagebus` image tag. The default value is `10.0.5.0-amd64`

Table 2. Configurable parameters (continued)	
Parameter name	**Description**
messagebus.image.testRepository	The utility image repository. Update this repository name to pull from a private image repository. The default value is `netcool-integration-util`
messagebus.image.testImageTag	Use this parameter to specify the utility image tag. The default value is `2.0.0-amd64`
messagebus.image.pullPolicy	Use this parameter to specify the image pull policy. The default value is `Always`
messagebus.netcool.connectionMode	The connection mode to use when connecting to the Netcool/OMNIbus Object Server. See Securing Probe and Object Server Communications section for more details. **Note :** Refer to the limitations section for more details on available connection modes for your environment. The default is `default`.
messagebus.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`
messagebus.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`
messagebus.netcool.primaryPort	Use this parameter to specify the port number of the primary Netcool/OMNIbus server (required). The default value is `nil`
messagebus.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`

Table 2. Configurable parameters (continued)	
Parameter name	**Description**
messagebus.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`
messagebus.netcool.backupPort	Use this parameter to specify the port of the backup Netcool/OMNIbus server. The default value is `nil`
messagebus.netcool.secretName	Use this parameter to specify a pre--created secret for AuthOnly, SSLOnly or SSLAndAuth connection modes. Certain fields are required depending on the connection mode. The default is `nil`.
messagebus.probe.messageLevel	Use this parameter to specify the probe log message level. The default value is `warn`
messagebus.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 `true`
messagebus.probe.transportType	Use this parameter to specify the probe transport type. The default value is KAFKA
messagebus.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`
messagebus.probe.secretName	Use this parameter to specify the Secret for authentication credentials for the HTTP and Kafka Transport. The default value is `nil`
messagebus.probe.rulesConfigmap	Use this parameter to specify a customised ConfigMap to be used for the rules files. This field may be left blank if not required. The default value is `nil`

Table 2. Configurable parameters (continued)	
Parameter name	**Description**
messagebus.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`
messagebus.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`
messagebus.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`
messagebus.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`
messagebus.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
messagebus.kafka.connection.zookeeperClient.target	Use this parameter to specify the ZooKeeper endpoint. The default value is `nil`
messagebus.kafka.connection.zookeeperClient.topicWatch	Use this parameter to enable the ZooKeeper topic watch service. The default value is `false`
messagebus.kafka.connection.zookeeperClient.brokerWatch	Use this parameter to enable the ZooKeeper broker watch service. The default value is `false`

Table 2. Configurable parameters (continued)	
Parameter name	**Description**
messagebus.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`
messagebus.kafka.connection.topics	Use this parameter to specify the topics in a comma-separated list, for example `topicABC,topicXYZ`. The default value is `nil`
messagebus.kafka.client.securityProtocol	Use this parameter to specify the security protocol. The default value is `nil`
messagebus.kafka.client.ssl.trustStoreSecretName	Use this parameter to specify the truststore Secret name and its password. The default value is `nil`
messagebus.kafka.client.ssl.keyStoreSecretName	Use this parameter to specify the keystore Secret name and its password. The default value is `nil`
messagebus.kafka.client.saslPlainMechanism	Use this parameter to enable the PLAIN mechanism for Simple Authentication and Security Layer connections. The default value is `false`
messagebus.kafka.client.consumer.groupId	Use this parameter to specify the group identifier. The default value is `nil`
messagebus.autoscaling.enabled	Use this parameter to enable or disable auto-scaling. The default value is `true`
messagebus.autoscaling.minReplicas	Use this parameter to specify the minimum number of probe replicas. The default value is 1
messagebus.autoscaling.maxReplicas	Use this parameter to specify the maximum number of probe replicas. The default value is 5

Table 2. Configurable parameters (continued)	
Parameter name	**Description**
messagebus.autoscaling.cpuUtil	Use this parameter to specify the target CPU utilization. For example, enter 60 for 60% target utilization. The default value is 60
messagebus.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
messagebus.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
messagebus.resources.limits.cpu	Use this parameter to specify the CPU resource limits. The default value is 500m
messagebus.resources.limits.memory	Use this parameter to specify the memory resource limits. The default value is 512Mi
messagebus.resources.requests.cpu	Use this parameter to specify the CPU resource requests. The default value is 250m
messagebus.resources.requests.memory	Use this parameter to specify the memory resource requests. The default value is 256Mi
messagebus.arch	Use this parameter to specify the worker node architecture. This is fixed to amd64.

Integrating the Message Bus Kafka probe with Apache Kafka

Configure the Message Bus Kafka probe to consume events from Apache Kafka using the following properties.

Configuring generic probe properties

Table 3. Generic probe properties

Property name	Configuring the property
Transport Type	Set to KAFKA.
Heartbeat Interval	Specifies the frequency (in seconds) with which the probe checks the status of the host server. Default is `10` seconds.
Custom Rules File ConfigMap Name	Specifies a customized ConfigMap to be used for the rules files. If this field is left blank, the default ConfigMap for the rules files will be used.
The following properties configure the probe's JSON parser to process the JSON event.	
Parser: JSON Message Payload (Consumer Events)	Specifies the endpoint name to be used by the probe Transport module. Multi-level JSON is specified in the following comma-separated format for example `json.message.alarm`. Default is `json`.
Parser: JSON Message Header (Consumer Events)	Specifies the JSON tree within a nested JSON or JSON string to be identified as message payload.
Parser: Nested JSON Payload (Consumer Events)	Specifies the JSON tree to be identified as message payload.
Parser: Nested JSON Header (Consumer Events)	Specifies the JSON tree within a nested JSON or JSON string to be identified as message header.
Parser: Message Depth (Consumer Events)	Specifies the number of levels in the message to traverse during parsing. Default is 3 levels.

Configuring Kafka probe properties

Table 4. Kafka probe properties

Property name	Configuring the property
ZooKeeper Client Target	Specifies the ZooKeeper endpoint for example `hostname:2181`. This property is mandatory and cannot be left blank.
Enable ZooKeeper Topic Watch Service	Enable the ZooKeeper topic watch service. Default is `false`.
Enable ZooKeeper Broker Watch Service	Enable the ZooKeeper broker watch service. Default is `false`.
Brokers	Specifies the broker endpoints in a comma-separated list, for example `localhost:9092, localhost:9093, localhost:9094`. This property is mandatory and cannot be left blank.

Table 4. Kafka probe properties (continued)	
Property name	**Configuring the property**
Topics	Specifies the topics in a comma-separated list, for example `topic1, topic2, topic3`. This property is mandatory and cannot be left blank.
Security Protocol	Configures the security protocol to be used as `PLAINTEXT`, `SASL_PLAINTEXT`, `SASL_SSL` or `SSL`.
SASL PLAIN Mechanism	Enable the PLAIN mechanism for Simple Authentication and Security Layer connections. Default is false.
Group Identifier	Specifies the Group Identifier assigned to the Message Bus Kafka probe as a Kafka consumer. The default is `test-consumer-group`.

Configuring Kubernetes secrets for sensitive information

The chart will read sensitive information (for example, authentication credentials and/or TrustStore/KeyStore files with their passwords) stored in Kubernetes Secrets.

Specify the Secrets to be used by the chart using the following properties:

Table 5. Kubernetes secrets probe properties	
Property name	**Configuring the property**
HTTP and Kafka Authentication Credentials Secret	Specifies the name of the Secret containing the authentication credentials for the HTTP and Kafka Transport.
Truststore Secret	Specifies the name of the Secret containing the TrustStore file and its password.
Keystore Secret	Specifies the name of the Secret containing the KeyStore file and its password.

To create the Secret for the `HTTP and Kafka Authentication Credentials Secret`, issue the following command:

```
kubectl create secret generic -n namespace kafka-auth
        --from-literal=http_username='httpuser'
        --from-literal=http_password='httppass'
        --from-literal=kafka_username='kafkauser'
        --from-literal=kafka_password='kafkapass'
```

where the chart namespace is `namespace`, the Secret name is `kafka-auth`, the HTTP Transport username is `httpuser`, the HTTP Transport password is `httppass`, the Kafka Transport username is `kafkauser` and the Kafka Transport password is `kafkapass`.

All literal fields must be specified, but the field content may be left blank if the credential is not to be used. Below is an example of the command with only the Kafka Transport credentials specified:

```
kubectl create secret generic -n namespace kafka-auth
        --from-literal=http_username=''
        --from-literal=http_password=''
        --from-literal=kafka_username='kafkauser'
        --from-literal=kafka_password='kafkapass'
```

To create the Secret for the `TrustStore Secret`, issue the following command:

```
kubectl create secret generic -n namespace kafka-trust
        --from-file=client-truststore.jks=client-truststore.jks
        --from-literal=truststore_password='password'
```

where the chart namespace is `namespace`, the Secret name is `kafka-trust`, the TrustStore file is `client-truststore.jks` and the TrustStore password is `password`.

To create the Secret for the `KeyStore Secret`, issue the following command:

```
kubectl create secret generic -n namespace kafka-keyst
        --from-file=client-keystore.jks=client-keystore.jks
        --from-literal=truststore_password='password'
```

where the chart namespace is `namespace`, the Secret name is `kafka-keyst`, the KeyStorre file is `client-keystore.jks` and the KeyStore password is `password`.

Example configuration

The following table shows an example of configuring the probe to connect to Apache Kafka using an SASL_SSL connection.

Table 6. Example configuration

Property name	Property value
HTTP and Kafka Authentication Credentials Secret	`kafka-auth`
ZooKeeper Client Target	`zookeeper:2181`
Enable ZooKeeper Topic Watch Service	`true`
Enable ZooKeeper Broker Watch Service	`true`
Brokers	`SASL_SSL://kafkaserver:9092`
Topics	`Topic1,Topic2`
Security Protocol	`SASL_SSL`
TrustStore Secret	`kafka-trust`
KeyStore Secret	`kafka-keyst`
SASL PLAIN Mechanism	`true`
Group Identifier	`test-consumer-group`

For chart installation using `helm install`, the following snippet of YAML applies the same configuration:

```
    # Probe Configuration
    probe:
        secretName: "kafka-auth"
```

```
# Kafka Configuration
kafka:
    connection:
        zookeeperClient:
            target: "zookeeper:2181"
            topicWatch: true
            brokerWatch: true
        brokers: "SASL_PLAINTEXT://kafkaserver:9092"
        topics: "Topic1,Topic2"
    client:
        securityProtocol: "SASL_SSL"
        ssl:
            trustStoreSecretName: "kafka-trust"
            keyStoreSecretName: "kafka-keyst"
        saslPlainMechanism: true
        consumer:
            groupId: "test-consumer-group"
```

Creating a custom rules file configmap

The rules files for the Message Bus probe can be customized and loaded with the helm chart. For details about creating a config map from a custom rules file see “Specifying a custom probe rules file” on page 29.

Integrating the Message Bus Kafka probe with Microsoft Azure Event Hubs for Kafka

Configure the Message Bus Kafka probe to consume events from Microsoft Azure Event Hubs for Kafka to consume events using the following properties.

Configuring generic probe properties

Table 7. Generic probe properties

Property name	**Configuring the property**
Transport Type	Set to KAFKA.
Heartbeat Interval	Specifies the frequency (in seconds) with which the probe checks the status of the host server. Default is 10 seconds.
Custom Rules File ConfigMap Name	Specifies a customized ConfigMap to be used for the rules files. If this field is left blank, the default ConfigMap for the rules files will be used.
The following properties configure the probe's JSON parser to process the JSON event.	
Parser: JSON Message Payload (Consumer Events)	Specifies the endpoint name to be used by the probe Transport module. Multi-level JSON is specified in the following comma-separated format for example `json.message.alarm`. Default is `json`.
Parser: JSON Message Header (Consumer Events)	Specifies JSON tree within a nested JSON or JSON string to be identified as message payload.
Parser: Nested JSON Payload (Consumer Events)	Specifies the JSON tree to be identified as message payload.

Table 7. Generic probe properties (continued)	
Property name	**Configuring the property**
Parser: Nested JSON Header (Consumer Events)	Specifies the JSON tree within a nested JSON or JSON string to be identified as message header.
Parser: Message Depth (Consumer Events)	Specifies the number of levels in the message to traverse during parsing. Default is 3 levels.

Configuring Kafka probe properties

Table 8. Kafka probe properties	
Property name	**Configuring the property**
ZooKeeper Client Target	As no Zookeeper endpoint exists for Microsoft Azure Event Hubs for Kafka, this property must be left blank.
Enable ZooKeeper Topic Watch Service	As no Zookeeper endpoint exists for Microsoft Azure Event Hubs for Kafka, this property must be set to `false`.
Enable ZooKeeper Broker Watch Service	As no Zookeeper endpoint exists for Microsoft Azure Event Hubs for Kafka, this property must be set to `false`.
Brokers	Specifies the broker endpoint, for example SASL_SSL://<namespace>.servicebus.windows.net:9093, where <namespace> is the Event Hubs namespace created to provide the Kafka service.. This property is mandatory and cannot be left blank.
Topics	Specifies the topic, for example `topic1`. This property is mandatory and cannot be left blank.
Security Protocol	Configures the security protocol to be used. Microsoft Azure Event Hubs for Kafka requires TLS for all communication. Set as SASL_SSL.
SASL PLAIN Mechanism	Enable the PLAIN mechanism for Simple Authentication and Security Layer connections. Set as `true`.
Group Identifier	Specifies the Group Identifier assigned to the Message Bus Kafka probe as a Kafka consumer. The default is `test-consumer-group`.

Configuring Kubernetes secrets for sensitive information

The chart will read sensitive information (for example, authentication credentials) stored in Kubernetes Secrets. Microsoft Azure Event Hubs for Kafka uses Shared Access Signatures (SAS) for authentication. You may obtain the 256-bit SAS keys from the Event Hubs namespace under `Shared Access Policies`. By default the Event Hubs namespace will contain the SAS key called `RootManageSharedAccessKey`.

Specify the Secrets to be used by the chart using the following properties:

Table 9. Kubernetes secrets probe properties

Property name	Configuring the property
HTTP and Kafka Authentication Credentials Secret	Specifies the name of the Secret containing the authentication credentials for the HTTP and Kafka Transport.

To create the Secret for the `HTTP and Kafka Authentication Credentials Secret`, issue the following command:

```
kubectl create secret generic -n namespace azure-auth
      --from-literal=http_username='$ConnectionString'
      --from-literal=http_password='<conn-string-pri-key>'
      --from-literal=kafka_username='$ConnectionString'
      --from-literal=kafka_password='<conn-string-pri-key>'
```

where the chart namespace is `namespace`, the Secret name is `kafka-auth` and `<conn-string-pri-key>` is the `Connection String-primary key` for the Azure Event Hubs namespace. The HTTP and Kafka Transport username must always be set to `$ConnectionString`.

Example configuration

The following table shows an example of configuring the probe to connect to Microsoft Azure Event Hubs for Kafka.

Table 10. Example configuration

Property name	Property value
HTTP and Kafka Authentication Credentials Secret	`azure-auth`
Brokers	`SASL_SSL://azure.servicebus.windows.net:9093`
Topics	`Topic1`
Security Protocol	`SASL_SSL`
SASL PLAIN Mechanism	`true`
Group Identifier	`test-consumer-group`

For chart installation using `helm install`, the following snippet of YAML applies the same configuration:

```
    # Probe Configuration
    probe:
```

```
        secretName: "azure-auth"

    # Kafka Configuration
    kafka:
        connection:
            zookeeperClient:
                target: ""
                topicWatch: false
                brokerWatch: false
            brokers: "SASL_SSL://azure.servicebus.windows.net:9093"
            topics: "Topic1"
        client:
            securityProtocol: "SASL_SSL"
            ssl:
                trustStoreSecretName: ""
                keyStoreSecretName: ""
            saslPlainMechanism: true
            consumer:
                groupId: "test-consumer-group"
```

Creating a custom rules file configmap

The rules files for the Message Bus probe can be customized and loaded with the helm chart. For details about creating a config map from a custom rules file see “Specifying a custom probe rules file” on page 29.

Integrating the Message Bus Kafka probe with IBM Event Streams

Configure the Message Bus Kafka probe to consume events from IBM Event Streams using the following properties.

Configuring generic probe properties

Table 11. Generic probe properties

Property name	**Configuring the property**
Transport Type	Set to KAFKA.
Heartbeat Interval	Specifies the frequency (in seconds) with which the probe checks the status of the host server. Default is 10 seconds.
Custom Rules File ConfigMap Name	Specifies a customized ConfigMap to be used for the rules files. If this field is left blank, the default ConfigMap for the rules files will be used.
The following properties configure the probe's JSON parser to process the JSON event.	
Parser: JSON Message Payload (Consumer Events)	Specifies the endpoint name to be used by the probe Transport module. Multi-level JSON is specified in the following comma-separated format for example `json.message.alarm`. Default is `json`.
Parser: JSON Message Header (Consumer Events)	Specifies JSON tree within a nested JSON or JSON string to be identified as message payload.
Parser: Nested JSON Payload (Consumer Events)	Specifies the JSON tree to be identified as message payload.

Table 11. Generic probe properties (continued)	
Property name	**Configuring the property**
Parser: Nested JSON Header (Consumer Events)	Specifies the JSON tree within a nested JSON or JSON string to be identified as message header.
Parser: Message Depth (Consumer Events)	Specifies the number of levels in the message to traverse during parsing. Default is 3 levels.

Configuring Kafka probe properties

Table 12. Kafka probe properties	
Property name	**Configuring the property**
ZooKeeper Client Target	ZooKeeper is not required for the probe to integrate with IBM Event Streams. This property must be left blank.
Enable ZooKeeper Topic Watch Service	As ZooKeeper is not required for the probe to integrate with IBM Event Streams, this property must be set to `false`.
Enable ZooKeeper Broker Watch Service	As ZooKeeper is not required for the probe to integrate with IBM Event Streams, this property must be set to `false`.
Brokers	Specifies the address of the bootstrap server which may be obtained from the IBM Event Streams UI, for example: `SASL_SSL://<bootstrap server>:<port>`.. This property is mandatory and cannot be left blank.
Topics	Specifies the topic, for example `topic1`. This property is mandatory and cannot be left blank.
Security Protocol	Configures the security protocol to be used. IBM Event Streams requires TLS for all communication. Set as `SASL_SSL`.
SASL PLAIN Mechanism	Enable the PLAIN mechanism for Simple Authentication and Security Layer connections. Set as `true`.
Group Identifier	Specifies the Group Identifier assigned to the Message Bus Kafka probe as a Kafka consumer. The default is `test-consumer-group`.

Configuring Kubernetes secrets for sensitive information

The chart will read sensitive information (for example, authentication credentials and/or the TrustStore file with its password) stored in Kubernetes Secrets.

Specify the Secrets to be used by the chart using the following properties:

Table 13. Kubernetes secrets probe properties

Property name	Configuring the property
HTTP and Kafka Authentication Credentials Secret	Specifies the name of the Secret containing the authentication credentials for the HTTP and Kafka Transport.
Truststore Secret	Specifies the name of the Secret containing the TrustStore file and its password. The TrustStore file can be downloaded from the IBM Event Streams UI.

To create the Secret for the `HTTP and Kafka Authentication Credentials Secret`, issue the following command:

```
kubectl create secret generic -n namespace evtstr-auth
      --from-literal=http_username='token'
      --from-literal=http_password='<token string>'
      --from-literal=kafka_username='token'
      --from-literal=kafka_password='<token string>'
```

where the chart namespace is `namespace`, the Secret name is `evtstr-auth`, the HTTP and Kafka Transport password is the token string obtained from the IBM Event Streams UI. The HTTP and Kafka Transport username must always be set to `token`.

To create the Secret for the `TrustStore Secret`, issue the following command:

```
kubectl create secret generic -n namespace evtstr-trust
      --from-file=client-truststore.jks=client-truststore.jks
      --from-literal=truststore_password='password'
```

where the chart namespace is `namespace`, the Secret name is `evtstr-trust`, the TrustStore file is `client-truststore.jks` and the TrustStore password is `password`.

Example configuration

The following table shows an example of configuring the probe to IBM Event Streams..

Table 14. Example configuration

Property name	Property value
HTTP and Kafka Authentication Credentials Secret	`evtstr-auth`
Brokers	`SASL_SSL://bootstrapserver:30186`
Topics	`Topic1`
Security Protocol	`SASL_SSL`

Table 14. Example configuration (continued)	
Property name	**Property value**
TrustStore Secret	evtstr-trust
SASL PLAIN Mechanism	true
Group Identifier	test-consumer-group

For chart installation using `helm install`, the following snippet of YAML applies the same configuration:

```
# Probe Configuration
probe:
    secretName: "evtstr-auth"

# Kafka Configuration
kafka:
    connection:
        zookeeperClient:
            target: ""
            topicWatch: false
            brokerWatch: false
        brokers: "SASL_PLAINTEXT://bootstrapserver:30186"
        topics: "Topic1"
    client:
        securityProtocol: "SASL_SSL"
        ssl:
            trustStoreSecretName: "evtstr-trust"
            keyStoreSecretName: ""
        saslPlainMechanism: true
        consumer:
            groupId: "test-consumer-group"
```

Creating a custom rules file configmap

The rules files for the Message Bus probe can be customized and loaded with the helm chart. For details about creating a config map from a custom rules file see “Specifying a custom probe rules file” on page 29.

Configuring the Message Bus Kafka probe to connect in secured mode

The Message Bus Kafka probe can be configured to use a secure connection to the Kafka server. When a secure connection is used, the probe connects to the Kafka server using the TLS protocol. The supported TLS versions are v1.2, v1.1 and v1.

The Message Bus Kafka probe provides HTTP and Kafka Transport functionality when integrating with the Kafka service. The probe can be configured to enable a secure connection for either or both the HTTP and Kafka Transport (subject to the Kafka service requirements).

Creating the Truststore and Keystore certificates

Use the following the steps to create the truststore and the keystore certificates for the Kafka server and the Message Bus Kafka probe. The example below uses the hostname kafkaserver for the Kafka server and creates the certificates in the JKS format.

1. Generate an SSL key and certificate for the Kafka broker:

   ```
   keytool -keystore server-keystore.jks -alias kafkaserver -validity 365 / -
   keyalg RSA -genkey
   ```

2. Create your own CA:

```
openssl req -new -x509 -keyout ca-key -out ca-cert -days 365 keytool -
keystore server-truststore.jks -alias CARoot -import -file ca-cert keytool -
keystore client-truststore.jks -alias CARoot -import -file ca-cert
```

3. Sign the certificate:

   ```
   keytool -keystore server-keystore.jks -alias kafkaserver -certreq -file
   cert-file openssl x509 -req -CA ca-cert -CAkey ca-key -in cert-file -out
   cert-signed / -days 365 -CAcreateserial -passin pass:password keytool -
   keystore server-keystore.jks -alias CARoot -import -file ca-cert keytool -
   keystore server-keystore.jks -alias kafkaserver -import -file cert-signed
   ```

   The Kafka server is to be configured to use `server-truststore.jks` and `server-keystore.jks` as the Truststore and Keystore certificates respectively.

Creating the Kubernetes Secrets for the Truststore and Keystore certificates

Next, create the Kubernetes Secrets for the certificates. It should be noted that the helm chart requires the certificate files to have the expected file names in order to work correctly.

1. Create the Kubernetes Secret for the client truststore with the name kafka-truststore. Ensure that the certificate file is named as `client-truststore.jks`.

   ```
   kubectl create secret generic -n namespace kafka-trust --from-file=client-
   truststore.jks=client-truststore.jks --from-
   literal=truststore_password='password'
   ```

2. Create the Kubernetes Secret for the client keystore with the name `kafka-keystore`. Ensure that the certificate file is named as `client-keystore.jks`. This step may be omitted if client authentication (two-way SSL authentication) is not enabled on the Kafka server.

   ```
   kubectl create secret generic -n namespace kafka-keyst --from-file=client-
   keystore.jks=client-keystore.jks --from-literal=keystore_password='password'
   ```

Configuring the Helm Chart to run using a Secure Connection

The Message Bus Kafka probe will enable a secure connection for the HTTP Transport when either one of the following conditions is met:

- `KeyStore Secret` specifies a Secret containing a KeyStore file and its password
- `HTTP and Kafka Authentication Credentials Secret` specifies a Secret containing the HTTP Transport authentication credentials and Security Protocol is specified as SASL_SSL or SSL

The Message Bus Kafka probe will enable a secure connection for the Kafka Transport when either one of the following conditions is met:

- `TrustStore Secret` specifies a Secret containing a TrustStore file and its password
- `HTTP and Kafka Authentication Credentials Secret` specifies a Secret containing the Kafka Transport authentication credentials and Security Protocol is specified as SASL_SSL or SSL

Configuring the Helm Chart to run using SASL_SSL

Use the following steps to configure the helm chart to run using the SASL_SSL option. The example uses `kafkaserver` and 9092 as the hostname and port respectively for the Kafka server.

1. Specify the Secret containing the HTTP and Kafka Transport Username and Password for `HTTP and Kafka Authentication Credentials Secret`.
2. The Broker should be specified in the following format: `SASL_SSL://kafkaserver:9092`
3. The Security Protocol must be set to SASL_SSL.
4. Specify the Secret containing the TrustStore file and its password for `TrustStore Secret`.

5. The following step is required only if client authentication (two-way SSL authentication) is enabled on the Kafka server. Otherwise, skip this step.

   Specify the Secret containing the KeyStore file and its password for `KeyStore Secret`.

   The Keystore Password is to be set to the password for the client keystore for example `password`.
6. Check SASL PLAIN Mechanism only if the SASL Plain mechanism is enabled on the Kafka server.

Configuring the Helm Chart to run using SSL

Use the following steps to configure for the helm chart to run using the SSL option. The example uses `kafkaserver` and `9092` as the hostname and port respectively for the Kafka server.

1. Specify the Secret containing the HTTP and Kafka Transport Username and Password for `HTTP and Kafka Authentication Credentials Secret`.
2. The Broker should be specified in the following format: `SSL://kafkaserver:9092`
3. The Security Protocol must be set to SSL.
4. Specify the Secret containing the TrustStore file and its password for `TrustStore Secret`.
5. The following step is required only if client authentication (two-way SSL authentication) is enabled on the Kafka server. Otherwise, skip this step.

   Specify the Secret containing the KeyStore file and its password for `KeyStore Secret`. Otherwise, skip this step and leave the Keystore Secret Name and Keystore Password empty.
6. Check SASL PLAIN Mechanism only if the SASL Plain mechanism is enabled on the Kafka server.

Specifying a custom probe rules file

The Message Bus Probe for Kafka Integration helm chart and the Message Bus Probe for Webhook Integration helm chart allow you to specify a custom probe rules file by overriding the default rules file config map.

By specifying an overriding config map, the custom probe rules config map is not managed by the helm release and will not be removed when the release is deleted. It can also be re-used by more than one Helm releases in the same namespace.

Creating An Overriding Rules Config Map

To create a new config map to override the default rules files config map, use the following steps:

1. Create a new custom rules file or re-use an existing rules file.
2. Name the rules file `message_bus.rules` or create a new copy of the existing rules file using the copy command as shown below.

   ```
   cp my_custom.rules message_bus.rules
   ```
3. Create a new config map from the `message_bus.rules` file because the expected config map key is `message_bus.rules`. This config map must be in the same namespace as the probe deployment. Optionally, specify the target namespace using the `--namespace <namespace>` option in the following command:

   ```
   kubectl create configmap mb-custom-rules \
   --from-file=message_bus.rules \
   --namespace default
   ```
4. Configure a new probe release and set the **`rulesConfigmap`** parameter to the config map name created in the step above (`mb-custom-rules`).

   Note : The full parameter path may differ between charts. For example, the Webhook Helm Chart uses `probe.rulesConfigmap` and the Kafka Helm Chart uses `messagebus.probe.rulesConfigmap`.

5. For UI installation, the **rulesConfigmap** parameter is labeled `Custom Rules File ConfigMap Name`.

Configuring the JSON parser

The Message Bus Probe JSON parser can be configured to process different JSON event structures according to the endpoint set by the probe's transport module. This is useful when the event source system sends different JSON data structure depending on the API. For example, some event sources supports alarm query for clients to pull or re-synchronize alarms will return a batch or array of alarms while notification events are usually sent individually.

The JSON parser has several configurable property in order to traverse the JSON event and extract data to create an event. The following table shows the configurable parameters of the parser:

Table 15. Configurable parser parameters

Parameter name	Description
endpoint	The endpoint name. This value must match the endpoint set by the probe's transport module and may vary between transport types. Some of the probe helm charts may not expose this parameter,and it is preconfigured in the helm chart template.
messagePayload	Specifies the JSON tree to be identified as message payload.
messageHeader	Specifies the JSON tree to be identified as a message header. Attributes from the headers will be added to the generated event.
jsonNestedPayload	Specifies JSON tree within a nested JSON or JSON string to be identified as message payload. `messagePayload` is used to specify the path to the attribute which value has the nested JSON string.
jsonNestedHeader	Similar to **jsonNestedPayload**, this parameter specifies the JSON tree within a nested JSON or JSON string to be identified as message header. `messagePayload` is used to specify the path to the attribute which value has the nested JSON string.
messageDepth	Specifies the number of levels in the message to traverse during parsing.

Parsing a JSON event with child nodes

The following sample shows a JSON event with child nodes.

```
{
    "message" : {
        "alarm": {
            "type":1,
            "title":"Bluemix Alert SEV2 - ibm.env5_syd.fabric.bosh.memoryPercent : st_bosh_mem_high memory percent > 93",
            "message":"Metrics: ibm.env5_syd.fabric.bosh.memoryPercent, Situation: st_bosh_mem_high ",
            "target":"ibm.env5_syd.fabric.bosh.memoryPercent",
            "situation":"st_bosh_mem_high",
            "isrecovered":false,
            "severity":2,
```

```
            "receivers":"administrator@mydomain.com",
            "timestamp":1481871659514
        }
    }
}
```

Given the sample JSON event as shown above, the parser can be configured to produce the following output for further rules files parsing and mapping to Object Server fields.

Table 16. Parser configuration for a JSON event with child nodes

Parser configuration	Parser output (key=value)
messagePayload= json.message.alarm	isrecovered=false message=Metrics: ibm.env5_syd.fabric.bosh.memoryPercent, Situation: st_bosh_mem_high receivers=administrator@mydomain.com resync_event=false severity=2 situation=st_bosh_mem_high target=ibm.env5_syd.fabric.bosh.memoryPercent timestamp=1481871659514 title=Bluemix Alert SEV2 - ibm.env5_syd.fabric.bosh.memoryPercent : st_bosh_mem_high memory percent > 93 type=1
messagePayload=json.message	alarm.isrecovered=false alarm.message=Metrics: ibm.env5_syd.fabric.bosh.memoryPercent, Situation: st_bosh_mem_high alarm.receivers=administrator@mydomain.com alarm.severity=2 alarm.situation=st_bosh_mem_high alarm.target=ibm.env5_syd.fabric.bosh.memoryPercent alarm.timestamp=1481871659514 alarm.title=Bluemix Alert SEV2 - ibm.env5_syd.fabric.bosh.memoryPercent : st_bosh_mem_high memory percent > 93 alarm.type=1 resync_event=false

Parsing a JSON event with an array

The sample JSON below contains a `data` attribute which is an array of data elements.

```
{
  "data": [
    {
      "id": "1222383102379613532",
      "type": "FilteredAlarm",
      "attributes": {
        "id": "1234567",
        "node-id": "node01",
        "state": "ACTIVE",
        "resource": "SWITCH",
        "condition-severity": "CRITICAL",
        "first-raise-time": "2017-07-06T13:43:19.000+0000",
        "last-raise-time": "2017-07-06T13:43:19.000+0000",
        "number-of-occurrences": 1,
        "acknowledge-state": "NOT_ACKNOWLEDGED"
        }
    },
    {
      "id": "-4067638085156070319",
      "type": "FilteredAlarm",
      "attributes": {
        "id": "7654321",
        "node-id": "node02",
        "state": "ACTIVE",
        "resource": "SWITCH",
        "condition-severity": "CRITICAL",
        "first-raise-time": "2017-07-06T13:42:59.000+0000",
        "last-raise-time": "2017-07-06T13:42:59.000+0000",
        "number-of-occurrences": 1,
```

```
        "acknowledge-state": "NOT_ACKNOWLEDGED"
        }
    }
  ]
}
```

Given the previous sample JSON event, the parser can be configured with the following.

Note : If the `messagePayload` is set to the parent attribute of the payload (json or the root), the array element will be indexed. When messagePayload is pointed to an array node, each array element becomes a single event.

Table 17. Parser configuration for a JSON event with an array

Parser configuration	Parser output (key=value)
messagePayload=json.data	attributes.acknowledge-state=NOT_ACKNOWLEDGED attributes.condition-severity=CRITICAL attributes.first-raise-time=2017-07-06T13:43:19.000+0000 attributes.id=1234567 attributes.last-raise-time=2017-07-06T13:43:19.000+0000 attributes.node-id=node01 attributes.number-of-occurrences=1 attributes.resource=SWITCH attributes.state=ACTIVE id=1222383102379613532 resync_event=false type=FilteredAlarm
messagePayload=json	data.0.attributes.acknowledge-state=NOT_ACKNOWLEDGED data.0.attributes.condition-severity=CRITICAL data.0.attributes.first-raise-time=2017-07-06T13:43:19.000+0000 data.0.attributes.id=1234567 data.0.attributes.last-raise-time=2017-07-06T13:43:19.000+0000 data.0.attributes.node-id=node01 data.0.attributes.number-of-occurrences=1 data.0.attributes.resource=SWITCH data.0.attributes.state=ACTIVE data.0.id=1222383102379613532 data.0.type=FilteredAlarm data.1.attributes.acknowledge-state=NOT_ACKNOWLEDGED data.1.attributes.condition-severity=CRITICAL data.1.attributes.first-raise-time=2017-07-06T13:42:59.000+0000 data.1.attributes.id=7654321 data.1.attributes.last-raise-time=2017-07-06T13:42:59.000+0000 data.1.attributes.node-id=node02 data.1.attributes.number-of-occurrences=1 data.1.attributes.resource=SWITCH data.1.attributes.state=ACTIVE data.1.id=-4067638085156070319 data.1.type=FilteredAlarm resync_event=false

Parsing a JSON event with a Nested JSON String

The following sample JSON contains a `value` attribute which has a nested JSON string.

```
{
  "payload": {
    "id": 1,
    "body": {
      "value": "{\"header\":{\"timestamp\":\"2017-07-06T09:07:56Z\",\"event\":{\"alarm\":{\"id\":\"1234567\",\"resource\":
       \"4\",\"node-id\":\"node01\",\"condition-type\":\"Link Down\",\"condition-severity\":\"WARNING\",\"condition-source\":
       \"NETWORK\",\"condition-state\":\"ACTIVE\",\"additional-text\":\"Link
       Down\",\"first-raise-time\":\"2000-01-01T00:00:58.000+0000\",\"last-raise-time\":\"2000-01-01T00:00:58.000+0000\",\
       "number-of-occurrences\":1,\"acknowledge-state\":\"ACKNOWLEDGED\",\"acknowledge-update-time\":
       \"2017-07-06T09:07:56.463+0000\",\"additional-attrs\":{\"source\":\"EMS-1\"}},\"_type\":\"alarmAcknowledged\"}}}",
      "attributes": 1
    }
  },
  "event": "sample_alarm"
}
```

The following table shows several configurations and the generated output of the probe parser for rules files parsing.

Note : The **messagePayload** parameter is set to the attribute which value contains a JSON string and **jsonNestedPayload** is set to the attribute within the JSON string to be treated as a payload to create an event.

Table 18. Parser configuration for a JSON event with a nested JSON string

Parser configuration	Parser output (key=value)
messagePayload=json.payload.body.value jsonNestedPayload=json.header.event.alarm	acknowledge-state=ACKNOWLEDGED acknowledge-update-time=2017-07-06T09:07:56.463+0000 additional-attrs.source=EMS-1 additional-text=Link Down condition-severity=WARNING condition-source=NETWORK condition-state=ACTIVE condition-type=Link Down first-raise-time=2000-01-01T00:00:58.000+0000 id=1234567 last-raise-time=2000-01-01T00:00:58.000+0000 node-id=node01 number-of-occurrences=1 resource=4 resync_event=false
messagePayload=json.payload.body.value jsonNestedPayload=json.header.event	type=alarmAcknowledged alarm.acknowledge-state=ACKNOWLEDGED alarm.acknowledge-update-time=2017-07-06T09:07:56.463+0000 alarm.additional-attrs.source=EMS-1 alarm.additional-text=Link Down alarm.condition-severity=WARNING alarm.condition-source=NETWORK alarm.condition-state=ACTIVE alarm.condition-type=Link Down alarm.first-raise-time=2000-01-01T00:00:58.000+0000 alarm.id=1234567 alarm.last-raise-time=2000-01-01T00:00:58.000+0000 alarm.node-id=node01 alarm.number-of-occurrences=1 alarm.resource=4 resync_event=false

Limitations

This solution has the following limitations:

- Only supports amd64 platform architecture.
- Only supports JSON event sources.
- When connecting to an Object Server in the same IBM Cloud Private cluster, you may connect the probe to the secure connection proxy which is deployed with the IBM Netcool Operations Insight chart to encrypt the communication using TLS but the TLS termination is done at the proxy. It is recommended to enable IPSec on IBM Cloud Private to secure cluster data network communications.

For details about the Message Bus Probe, see https://www.ibm.com/support/knowledgecenter/en/SSSHTQ/omnibus/probes/message_bus/wip/concept/messbuspr_intro.html.

Troubleshooting

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

Table 19. Problems

Problem	Cause	Resolution
Pods failed to start because of a missing Key.	A pod volume mount failed with `"MountVolume.SetUp failed for volume "kafka-client-truststore" : references non-existent secret key"`	The pre-created secret must have the expected keys. See “Configuring the Message Bus Kafka probe to connect in secured mode” on page 27.
Pod logs error `"Not reviving"`, then unsubscribes the Kafka Consumer and exits.	Incorrect configuration of the Message Bus for Kafka Probe	Verify and correct all Kafka properties, for example: **`Brokers`**, **`Username`**, **`Password`**, and so forth. before deploying the probe chart again.

Appendix A. Notices and Trademarks

This appendix contains the following sections:

- Notices
- Trademarks

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing 2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Software Interoperability Coordinator, Department 49XA

3605 Highway 52 N
Rochester, MN 55901
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. _enter the year or years_. All rights reserved.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

IBM, the IBM logo, ibm.com, AIX, Tivoli, zSeries, and Netcool are trademarks of International Business Machines Corporation in the United States, other countries, or both.

Adobe, Acrobat, Portable Document Format (PDF), PostScript, and all Adobe-based trademarks are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, other countries, or both.

Intel, Intel Inside (logos), MMX, and Pentium are trademarks of Intel Corporation in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.

SC27-9524-02