Unit 2. Incoming integrations

Estimated time

00:45

Overview

This unit describes two ways to connect Event Manager to data sources: probes and webhooks.

References

About probes:

https://www.ibm.com/docs/SSSHTQ_8.1.0/pdf/omn_pdf_prgw_master.pdf

https://www.ibm.com/docs/netcoolomnibus/8.1?topic=gateways-setting-up-probes-acquire-event-data

https://www.ibm.com/docs/noi/1.6.7?topic=sources-connecting-event-cloud-deployment

About webhooks:

https://www.ibm.com/docs/noi/1.6.7?topic=systems-configuring-incoming-integrations

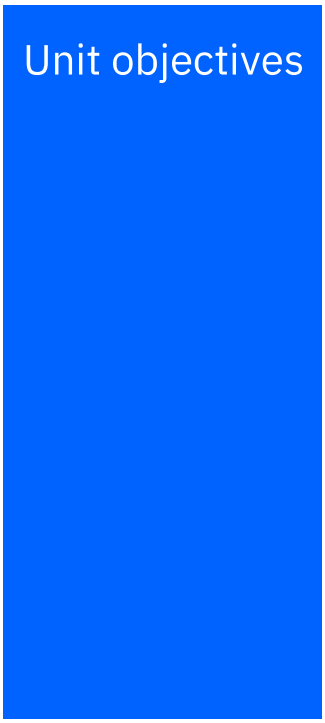


Figure 2-1. Unit objectives

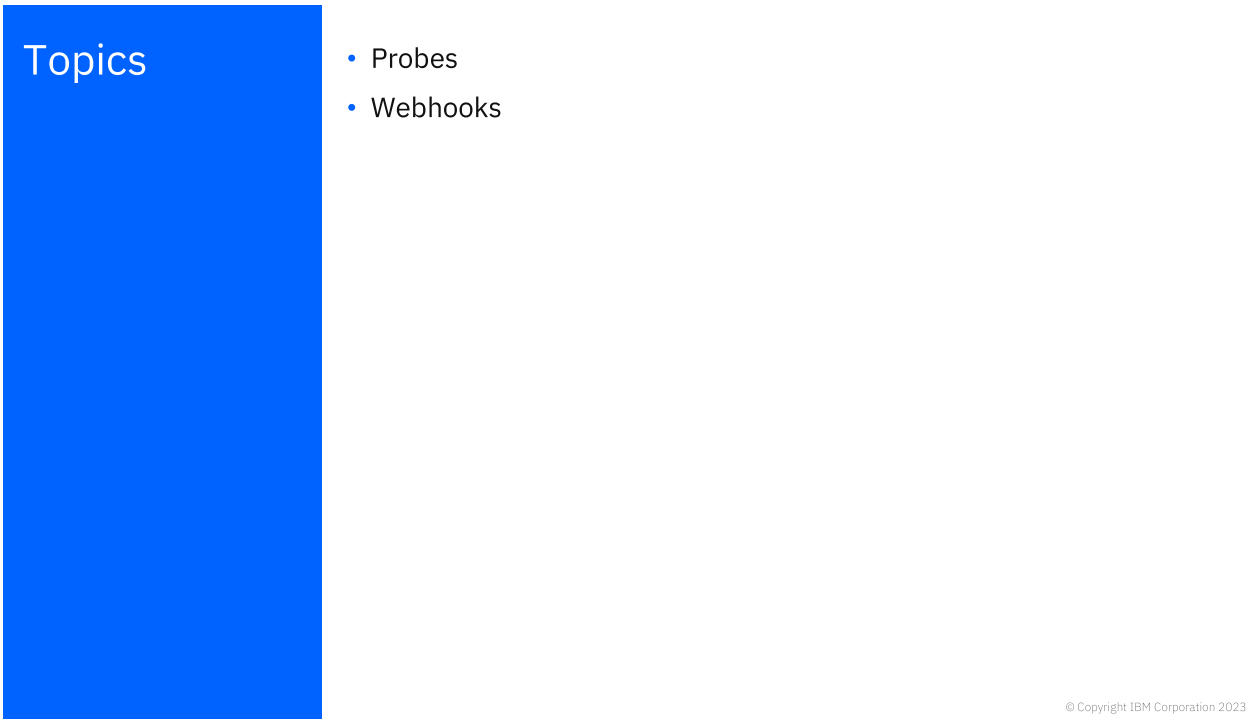


Figure 2-2. Topics

2.1. Probes

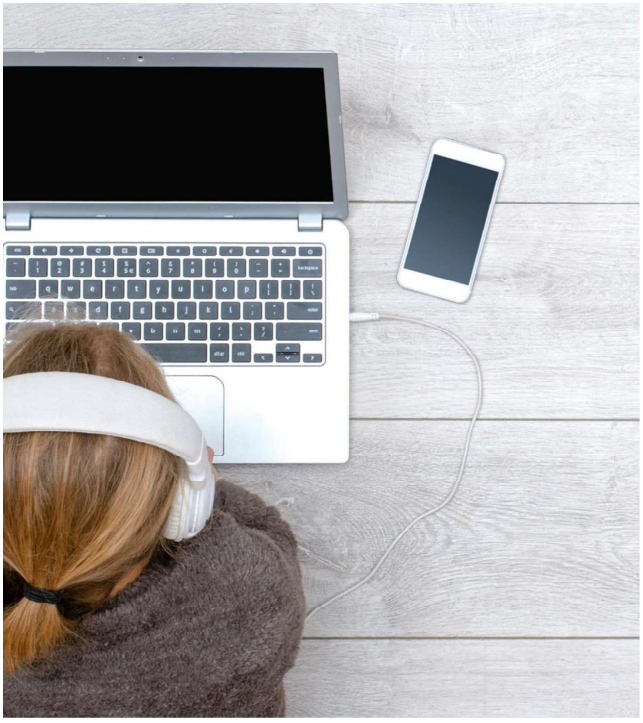
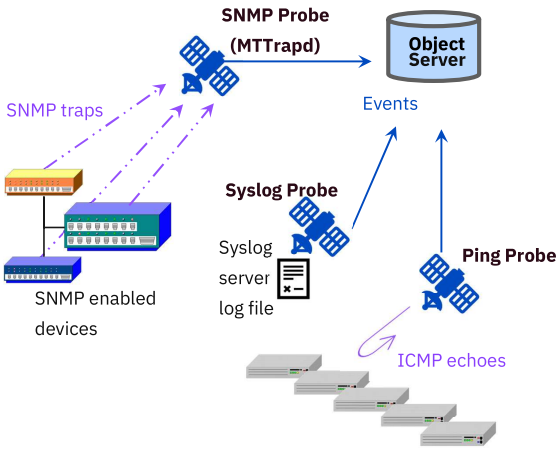


Figure 2-3. Probes

About probes

Probes:

- Are software data collectors
- Are designed to collect data from specific sources
  - Log files
  - Database tables
  - SNMP traps
  - Applications
  - Many others
- Normalize data to common format
- Generate ObjectServer events
- Are lightweight
- High performance
- Are resilient



The IBM Cloud Pak for Watson AIOps includes a large library of prebuilt probes.

Figure 2-4. About probes

Probes are light-weight software designed to collect data from a specific source and produce ObjectServer events.

Typical questions regarding probes:

- **What probes are required?**

  It depends upon event source. For example, devices that generate SNMP traps use the MTTrapd probe. Devices that generate log messages use the Syslog probe.

- **How many probes are required?**

  It depends upon the event source. For example, one MTTrapd Probe can receive traps from lots of devices. You require one Syslog Probe for every UNIX log file, because the probe physically reads one file.

- **Where do the probes run?**

  It depends upon the event source. For example, the MTTrapd probe can run any place where it can receive SNMP traps. The Syslog probe must run on the server with the UNIX log file.

- **What operating system is required by the probe?**

  It varies by the probe. For example, the MTTrapd probe is supported on AIX, Solaris, HP-UX, Linux, and Windows. The Syslog probe is not supported on windows because it reads a UNIX log file.

In this example, there are three probes collecting data and sending events to the ObjectServer:

- **The MTTrapd probe** listens for incoming SNMP traps sent by SNMP-enabled devices. The probe converts the OID in the SNMP traps into events, then sends the events to the ObjectServer.

- **The Syslog probe** watches a log file. When an interesting message is written to the log, the probe converts the fields in the message into an event, then sends the event to the ObjectServer.
- **The Ping probe** reads a list of IP addresses or host names, then performs a ping sweep of the resources in the list. The probe converts failed ICMP ping responses into events, then sends the events to the ObjectServer.

Types of probes

Alcatel-Lucent 1300 XMC
Alcatel-Lucent 5529 OAD V6
Alcatel-Lucent 5620 Logfile
Alcatel-Lucent 5620 SAM 3GGP v8
Alcatel-Lucent 5620 SAM v13 JMS
Alcatel-Lucent 5ESS
Alcatel-Lucent 9353 WNMS
Alcatel-Lucent DSC DEX
Alcatel-Lucent ECP
Alcatel-Lucent ITM-NM/OMS
Alcatel-Lucent ITM-SC
Alcatel-Lucent OMC-R
Alcatel-Lucent OS-OS
Alcatel-Lucent Wavestar SNMS (CORBA)
Amazon Web Services
Apache Pulsar
Avaya Definity G3
BMC Patrol
BMC Patrol V9
CA Spectrum V9 CORBA
CA Spectrum V9.4 (CORBA)
Ciena Blue Planet MCP
Cisco APIC
Cisco Evolved Programmable Network Manager
Cisco Transport Manager 9.0 (CORBA)
Comverse TRILOGUE INfinity
Dantel PointMaster
ECI Network Manager
Email
Exec
FIFO
Genband IEMS
Generic 3GPP
Generic Log File Java
Generic Multi-Technology Operations Systems Interface (MTOSI)
Generic TMF814
Glenayre VMS
Heartbeat
HP Network Node Manager-i
HP Operations Manager
HPE Operations Manager i
HTTP Server Error Log
Huawei M2000 MML
Huawei U2000 3GPP (CORBA)
Huawei U2000
IBM Event Streams for IBM Cloud
IBM SevOne Network Performance Management (NPM)
IBM Tivoli Common Event Infrastructure (CEI)
IBM Tivoli EIF
IBM Turbonomic
IBM WebSphere MQ
iDirect Pulse
IEC CIM Advanced Metering Infrastructure
Itron OpenWay Collection Engine (OWCE) EMS
JDBC
Juniper Contrail
Juniper Contrail Alerts
Kafka

And many more!

Incoming integrations

Figure 2-5. Types of probes

This slide shows a partial list of IBM Cloud Pak for Watson AIOps probes. There are over 100 different types of probes to support the diverse technologies found in modern IT environments. New or updated probes are released on a regular basis.

Probe operation

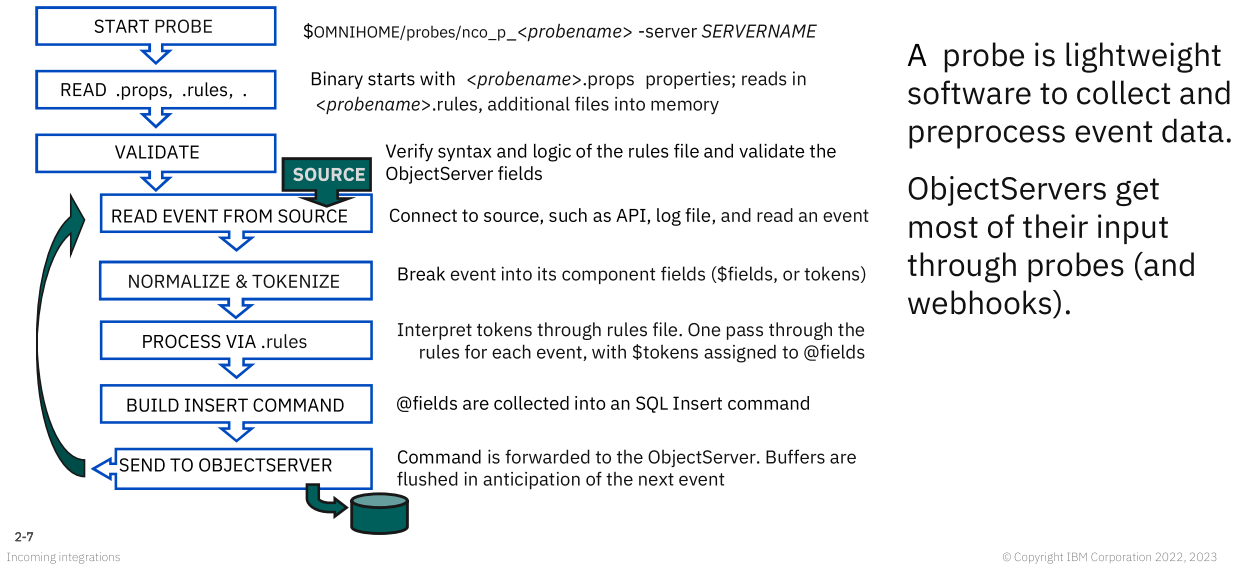


Figure 2-6. Probe operation

A probe is a light-weight, small-footprint software that obtains event data, converts it to a common event format (CEF) and passes it to the ObjectServer.

Probes are separate from the ObjectServer. As requirements change, probes can be added or removed without changing the ObjectServer or interrupting service. Currently probes are available for over 100 types of element managers, devices, and systems. Some probes are generic (SNMP MTTrapd, Syslog) while others are specific to applications or devices.

Probes use a reliable TCP connection to the ObjectServer to ensure completeness and accuracy of data. If a probe loses contact with the ObjectServer it can store events until the ObjectServer becomes available. When there are two ObjectServers, a probe can be configured to fail over, sending events to the second ObjectServer.

Probe operation can be split into five stages:

Table 1. Five stages of probe operation

Stage	Description
Initialize	The probe connects to the ObjectServer, identifying the format of the alerts.status table. The props and rules files are read into memory and parsed, and the probe is ready to retrieve events.
Event Retrieve	The probe retrieves an event from the source, for example from an API, SNMP trap, reading a log file, or other.
Tokenize	The probe tokenizes the event stream to create tokens ($fields) which are used in the rules file.

Table 1. Five stages of probe operation

Stage	Description
Process	The tokenized event stream is parsed through the rules file and ObjectServer alerts.status fields (@ fields) are set.
Forward	The composed event is forwarded to the ObjectServer. If problems occur in forwarding, the probe might fail over or store and forward.

After forwarding, the probe clears variables, retrieves a new event, and repeats the cycle as necessary.

Probe Basics

A probe consists of a **binary**, a **.rules** and a **.props** file:
- Binaries retrieve and tokenize event streams($OMNIHOME/probes/nco_p_<*probename*>)
- Properties run time settings of probe ($OMNIHOME/probes/<arch>/<*probename*>.props)
- Rules instructions for processing event ($OMNIHOME/probes/<arch>/<*probename*>.rules)

Probes can have additional files in **$OMNIHOME/probes/<arch>/**

If the probe is on a separate machine from the ObjectServer
- Install common components, process control
- Install the probe itself
- Identify the ObjectServers in the interfaces file

```
OBJ_SERV        hostname              4100
```

- Edit the properties and rules files of the probe
- Run the probe

```
$OMNIHOME/probes/nco_p_probename
```

Incoming integrations

Figure 2-7. Probe Basics

All probes have at least three files: a binary executable (nco_p_probename), an interpreted rules file (probename.rules) and a properties (probename.props) file.

The properties file sets run time parameters and determines the behavior of a probe. This includes, among other parameters, where to record a log file, and how verbose the log file messages should be.

The binary collects the event stream and splits it into individual tokens. The binary interprets and applies the rules file and forwards the processed event to the Object Server.

The primary purpose of the rules file is to assign tokens to Object Server fields. The rules file can also manipulate data, perform calculations, and derive additional data and add it to the event. The rules file can derive additional data using lookup tables and other methods.

When installing probes on a remote machine, you must install the probes and common files on that machine. These common files are installed when you install the probe.

Probes require access to the interfaces file to determine how to communicate with the ObjectServer. Use the nco_xigen or nco_igen utility to define the Object Server in the interfaces file.

Modify the properties and rules file as necessary and start the probe using the following command:

```
$OMNIHOME/probes/nco_p_probename
```

The following example shows the $OMNIHOME/probes/<arch>/ directory of a machine that is running the Simnet and Syslog probes.

```
$ pwd
/opt/IBM/tivoli/netcool/omnibus/probes/linux2x86

$ ls -l
total 232
netcool ncoadmin    144 default
netcool ncoadmin  27100 nco_p_simnet
netcool ncoadmin  18772 nco_p_syntax
netcool ncoadmin 137970 nco_p_syslog
netcool ncoadmin   1220 simnet.def
netcool ncoadmin   4007 simnet.props
netcool ncoadmin   1496 simnet.rules
netcool ncoadmin   2234 syntax.props
netcool ncoadmin    528 syntax.rules
netcool ncoadmin   2451 syslog.props
netcool ncoadmin  23337 syslog.rules
```

Rules file

- Contains program steps, which are run for each event to manipulate incoming data and assign it to alerts.status fields
- Found in $OMNIHOME/probes/<arch>/<probename>.rules
- Inbound tokens start with dollar sign (**$**); ObjectServer fields start with **@**
- Major function of rules file is to define Identifier field
- Field values of alerts.status might be set by the rules file
- Additional information can be added through the rules file
- Probe can have multiple associated rules files (include files)

Figure 2-8. Rules file

Not all data contained in an event is relevant to the processing of that event. The rules file defines how the probe rationalizes or adds to the contents of an incoming event to create a meaningful alert.

An important function of the rules file is to define the Identifier field used for de-duplication. If the Identifier is made too specific, for example by incorporating the time of the event, little de-duplication takes place. However, if the identifier is not specific enough, for example by omitting a card, port or slot number, wrong events are de-duplicated.

All field values of the alerts.status table are normally propagated by the rules file using the information from the incoming event. It is also possible to add extra information to the event in the rules file, such as customer, department, and application data.

Rules files can selectively update fields in alerts.status, overriding ObjectServer de-duplication settings.

The probe rules file is where the incoming token stream is parsed and assigned to ObjectServer fields.

It is a best practice to implement as much functionality as possible at the probe rules file level, before events are sent to the ObjectServer. This means the ObjectServer has less work to do when the event arrives. For example, dropping events that can be discarded is best done at the probe rules file level.

Rules file example

$OMNIHOME/probes/<arch>/simnet.rules

```
@Node=$Node
@Summary=$Summary
 switch($Severity) {
        case Critical:
           @Severity=5
        case Major:
           @Severity=4
        default:
           @Severity=2
 }
if (regmatch(@Summary,interface.*down)) {
      @AlertKey=extract(@Summary,interface (.*) is)
}
@Identifier = @Node + @AlertKey + @Summary
```


Figure 2-9. Rules file example

A simple rules example:

- Set Node and Summary field values using a direct assignment.
- Set the `Severity` field based on the value in the `$Severity` token. Note, the value in `$Severity` is a ext value while `@Severity` must be an integer.
- Extract the interface value from the `Summary` field and set the `AlertKey` field.
- Set the `Identifier` to ensure that this event is unique.

Note

The example on the slide is not a complete rules file.

Tokens and fields

- Probes split the event stream into tokens
- All tokens are created as strings
- Tokens in the rules file are denoted with a **$** prefix
- Example: **$Node** is a token holding the Node name
- Tokens can be created immediately: **$MyElement=somevalue**
- Fields in **alerts.status** are denoted with an **@** prefix
- To populate fields, you can assign values in these ways:
  - Direct assignment: **@Node = $Node**
  - Concatenation: **@Summary = $Summary + $group**
  - Concatenation with literals:

@Summary = $Node + has problem + $Summary

Figure 2-10. Tokens and fields

When a probe receives an event stream from the source, it splits the stream into tokens. The set of tokens is not fixed and might change depending on event data received. The tokens are identified in the rules file by `$`, for example `$Node` is a token holding the node name.

In the probe rules file, an ObjectServer field value is denoted with a `@`. For example, `@Node` references the value of the `Node` field. It is the @fieldnames in the alerts.status table of the ObjectServer which make up an event and are shown in event lists. To populate the @fieldnames, tokens need to be assigned to them, for example:

- Direct Assignment: `@Node=$Node`
- Concatenation: `@Summary=$Summary + $Group`
- Adding text: `@Summary=$Node + " has problem " + $Summary`

Testing and logic in rules files

Tests for string values:

```
match(@Node,router1)            Exact match
nmatch(@Node,router)            Begins with
regmatch(@Node,^router[0-9])    Full regex matching
```

`if` statement incorporates conditional logic:

```
if ( <test1> )
{  <action>  }
else if ( <test2> )
{  <action>  }
else
{  <action>  }
```

Example: set Class field based on the value of `$EquipType`

```
if (match($EquipType,Router)) {
      @Class = 3303
}
else if (nmatch($EquipType,Switch)) {
   @Class = 3301
}
else if (regmatch($EquipType,^[Hh]ub.*)){
    @Class = 3302
}
```


Figure 2-11. Testing and logic in rules files

String values can be tested with the following functions:

- Exact match: $Node = “router1”.

  ```
  match($Node,"router1")
  ```

- Begins with: $Node must begin with “router” but can have other text following.

  ```
  nmatch($Node,"router")
  ```

- Regex match: Allows full regular expressions. $Node must begin with “router” followed by a digit.

  ```
  regmatch($Node,"^router[0-9]")
  ```

These functions can be used in `if`, `else if`, `else` statements.

The `if` statement provides conditional testing for processing elements in rules files.

In the example in the slide, a <condition> is a combination of expressions and operations that resolve to either `TRUE` or `FALSE`. If the <condition> evaluates to `TRUE`, the <action> statements between the curly braces are run.

All comparisons can be used in the `if` statement. The `if` statement is typically used to conditionally set `@Field` values depending on the event received.

The example shown uses the `$EquipType` token to set the `Class` value. Events with equipment type `Router` have their `Class` set to `3303`. Events with equipment type beginning with `Switch` have `Class` set to `3301`. Events with equipment type beginning with upper or lowercase `H`, hub have their `Class` value set to `3302`.

When using an @Fieldname in an `if` statement, make sure that its value is already set in an assignment statement.

Lookup tables

- Technique for performing event enrichment
  - Adding data to the event record that does not appear in the probe source
- Data is formatted in a text file
  - Tab delimited
- File contains multiple columns
  - Key
  - Data item 1
  - Data item 2
  - Data item *N*
- Suitable for static data
  - Building names
  - Addresses

Figure 2-12. Lookup tables

Lookup tables provide a method of making extra information available to a probe, and inclusion in an event.

Lookup files are useful at the probe level for basic event enrichment so long as the data set is fairly static. If the data in the lookup table is dynamic, it is better to store it in a database table (for example, the ObjectServer or other external database) and then perform event enrichment with an ObjectServer trigger or with Impact.

A lookup table can be defined two ways. One way is a reference to an external file containing the table. The second way is by placing the table in the rules file itself.

For an external file table, you specify a pointer to the file table as follows:

```
table TabNam="/opt/netcool/omnibus/probes/<arch>/file"
```

This reference must be at the top of the rules file, as the first uncommented line above the `ProbeWatch` section.

The file must have the following format:

```
key[TAB]value
key[TAB]value
```

For a lookup table embedded in the rules file, the definition takes the format:

```
table TabNam={{"key","value"},{"key","value"}…}
```

In the rules file, the `lookup()` function looks the same for both table types. It uses two arguments: a value to look up and the name of the table to read from:

```
@result=lookup(@Key,TabNam)
```

Lookup table example

Rules file excerpt

```
...output omitted...
table labtable = "/opt/IBM/tivoli/netcool/omnibus/probes/linux2x86/labExample.lookup"
default = {"UNKNOWN","UNKNOWN"}

...output omitted...

     [ @Location,@Customer ] = lookup(@Node, labtable)
         update(@Location)
         update(@Customer)
}
```

labExample.lookup

```
interfaceEth0_1     LONDON    Bureau of Machine Analytics
interfaceEth0_2     BRUSSELS  Department of Computer Algorithms
interfaceFe2_1      SYDNEY    Department of Computer Algorithms
interfaceEth1_3     ROME      Oaca Industries, JP
```


Figure 2-13. Lookup table example

In this example, imagine you work for a communications provider. Part of your business is to provide managed network services to your customers. Each of your customers has their own managed network interfaces connected to your equipment, which you are monitoring with Event Manager. The goal in this example is to enrich incoming events:

- To add the name of the customer each interface is assigned to
- To add the location

This slide shows an excerpt from a rules file and a lookup table.

About the rules file example

There are two lines at the top of the rules file that define the table. The first line defines a lookup table named `labtable`. The second line provides the default values of `UNKNOWN` for the `@Location` and `@Customer` fields if no business data is found in the lookup table for a node.

There are three lines at the bottom of the rules file that use the lookup table named `labtable`. These lines use the lookup table to match the value of `@Node`. If a match is found in the table, the probe populates the `@Location` and `@Customer` fields in an event before it is sent to the Event Manager ObjectServer.

In this example, the `@Node` field is populated with interface names, although that assignment is not shown in the excerpt.

About the example lookup table

The table is in a tab-delimited text file named labExample.lookup. The first column in the file is a list of node names. In this example, the nodes in the incoming event are interface names. The

other two columns list the location and customer for each interface. There is nothing special about the lookup table file, except that it:

- Must be delimited with tabs.
- Must store the table key in the first column.
- Must be saved in a location and with permissions so that the probe can read it.

Netcool Knowledge Library (NcKL)

- The default rules file necessary for the execution of a probe only performs generic grouping of data.
- Using an enhanced rules file to cater to events from a specific device provides sharpened event enrichment and causal analysis.
- The library is a collection of rules files written to a common standard, and provides unprecedented levels of event correlation and causal analysis.

Figure 2-14. Netcool Knowledge Library (NcKL)

he Netcool Knowledge Library (NcKL) is a collection of rules files. It is distributed with Event Manager as a separate download.

This library of rules files:

- Uses the `include` technique to incorporate multiple individual files
  - One “leader” rules file contains multiple `include` statements
- Predefined rules files for multiple vendors and technologies
  - Packaged as individual files (over 2000 in current version)
  - Easy to configure to add or remove
- Rules provided for MTTrapd and Syslog probes

To install the Netcool Knowledge Library:

- Unpack rules files into target directory - $NC_RULES_HOME
- Run supplied SQL file to add ObjectServer customizations
- Configure probe to use the NcKL “leader” rules file

```
$NC_RULES_HOME/syslog.rules – Syslog Probe
$NC_RULES_HOME/snmptrap.rules – MTTrapd Probe
```

2.2. Webhooks

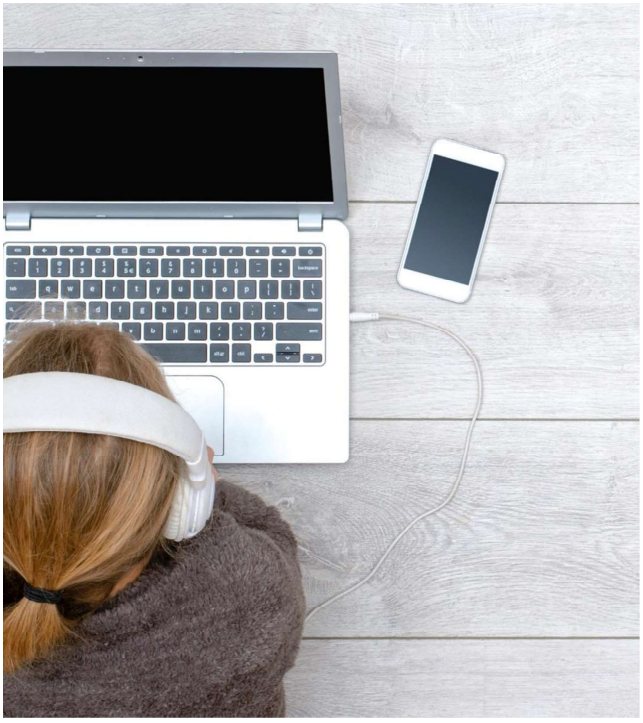


Figure 2-15. Webhooks

About inbound webhooks

- A webhook is a mechanism that allows you to send events to the IBM Cloud Pak for Watson AIOps using a REST-like API.
- You use webhooks to send events from other software systems to the IBM Cloud Pak for Watson AIOps.
- The payload of the incoming events must be formatted in JSON.

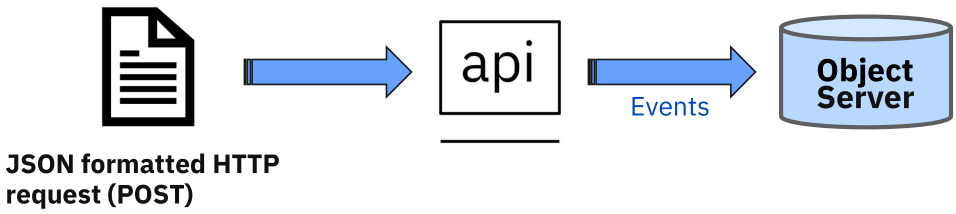


Figure 2-16. About inbound webhooks

Probes are not the only way to send events to the IBM Cloud Pak for Watson AIOps. You can use a webhook to send events to an API, which then normalizes the incoming data into an event record.

Data sent to a webhook must be in a JSON-formatted HTTP request.

Incoming integrations

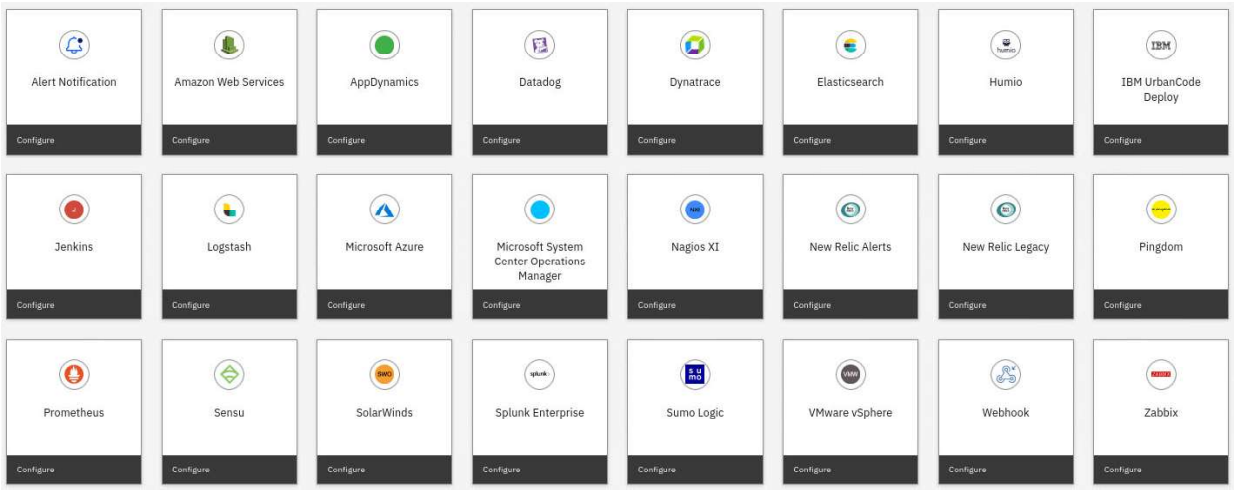


Figure 2-17. Incoming integrations

Several applications and platforms can send event data to a webhook API. The IBM Cloud Pak for Watson AIOps has predefined webhook integrations that already know how to map incoming HTTP request fields into ObjectServer fields. There are many predefined webhook integrations to ingest data from diverse sources, such as Amazon's Simple Notification Service (SNS), Microsoft Azure, Jenkins, and many more.

This slide shows the Incoming integrations page where you can configure connections to other systems. Not all of the integrations shown on this slide use webhooks to accept data, but most of them do.

Custom inbound webhooks

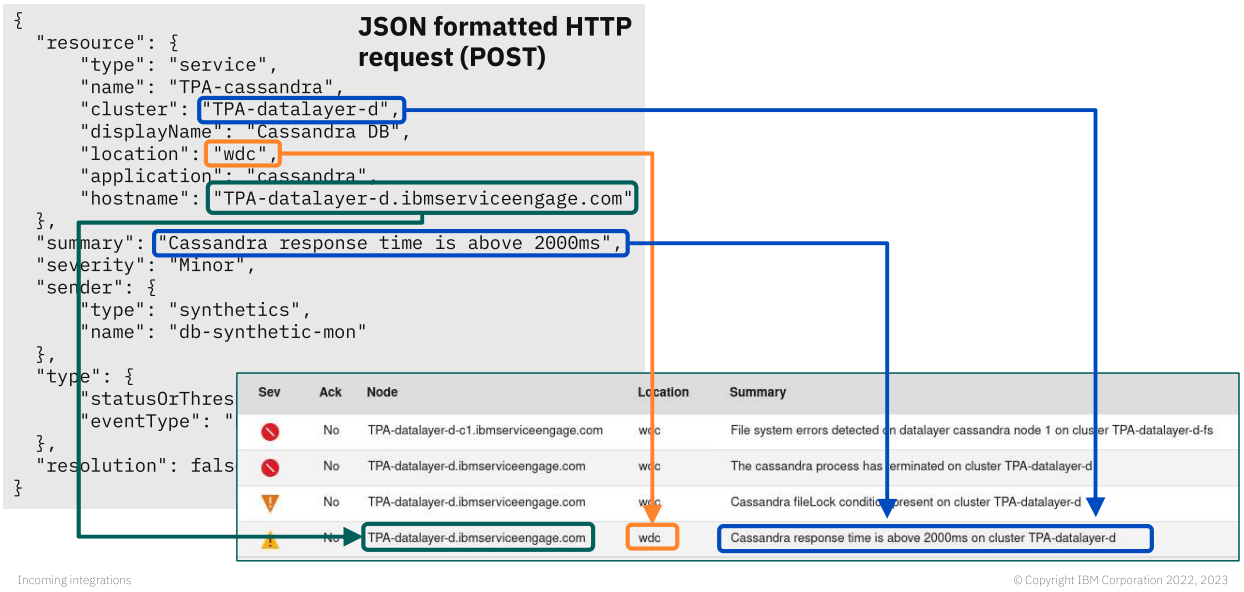


Figure 2-18. Custom inbound webhooks

Even if there is no predefined integration for a data source in your environment, you can send data to an Event Manager webhook. You can create a custom webhook to map the fields within any incoming JSON HTTP request to an event record.

In this example, a custom webhook extracts text from an incoming HTTP request and uses the text to populate an event record:

- The **`resource.hostname`** field in the incoming HTTP request is mapped to the **`Node`** field of the ObjectServer.
- The **`resource.location`** field in the incoming HTTP request is mapped to the **`Location`** field of the ObjectServer.
- The **`summary`** and **`resource.cluster`** fields in the incoming HTTP request are combined with some static text, and mapped to the **`Summary`** field of the ObjectServer.
- The **`severity`** field in the incoming HTTP request is mapped to the **`Severity`** field of the ObjectServer.

IBM Cloud Pak for Waston AIOps Event Manager provides an easy to use tool to create custom inbound webhooks.

Unit summary

- Understand how probes collect and process data
- Learn how to connect an on-premises probe to cloud-based Event Manager
- Describe how to use probes to enrich events
- Learn how to send events using an inbound webhook

Figure 2-19. Unit summary

Review questions

1. True or False: Probes are the only way to send events to the IBM Cloud Pak for Watson AIOps.
2. What is a probe rules file?
   a. A configuration file that controls how a probe creates an event from incoming data.
   b. A configuration file that controls how a probe handles failover and recovery.
   c. A checkpoint file that saves the current state of the probe to disk.
   d. A checkpoint file that saves all outbound events to disk.

Figure 2-20. Review questions

Write your answers here:

1.

2.

Review answers

1. True or False: Probes are the only way to send events to the IBM Cloud Pak for Watson AIOps.
   The answer is False.
2. What is a probe rules file?
   a. A configuration file that controls how a probe creates an event from incoming data.
   b. A configuration file that controls how a probe handles failover and recovery.
   c. A checkpoint file that saves the current state of the probe to disk.
   d. A checkpoint file that saves all outbound events to disk.
   The answer is A.

Figure 2-21. Review answers

Exercise: Incoming integrations

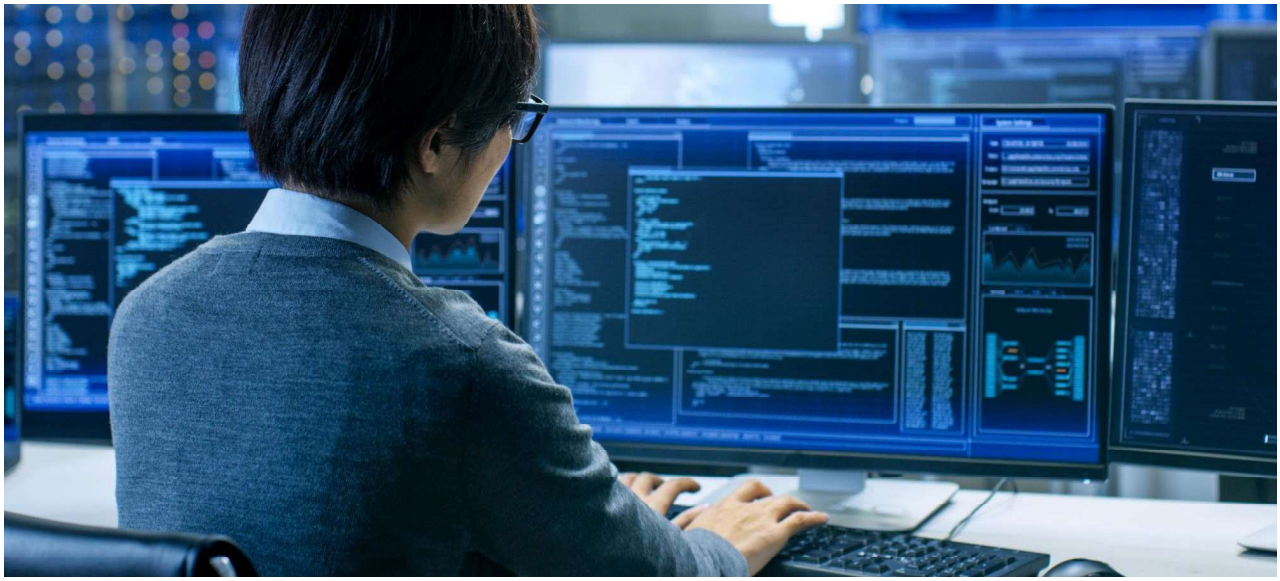


Figure 2-22. Exercise: Incoming integrations

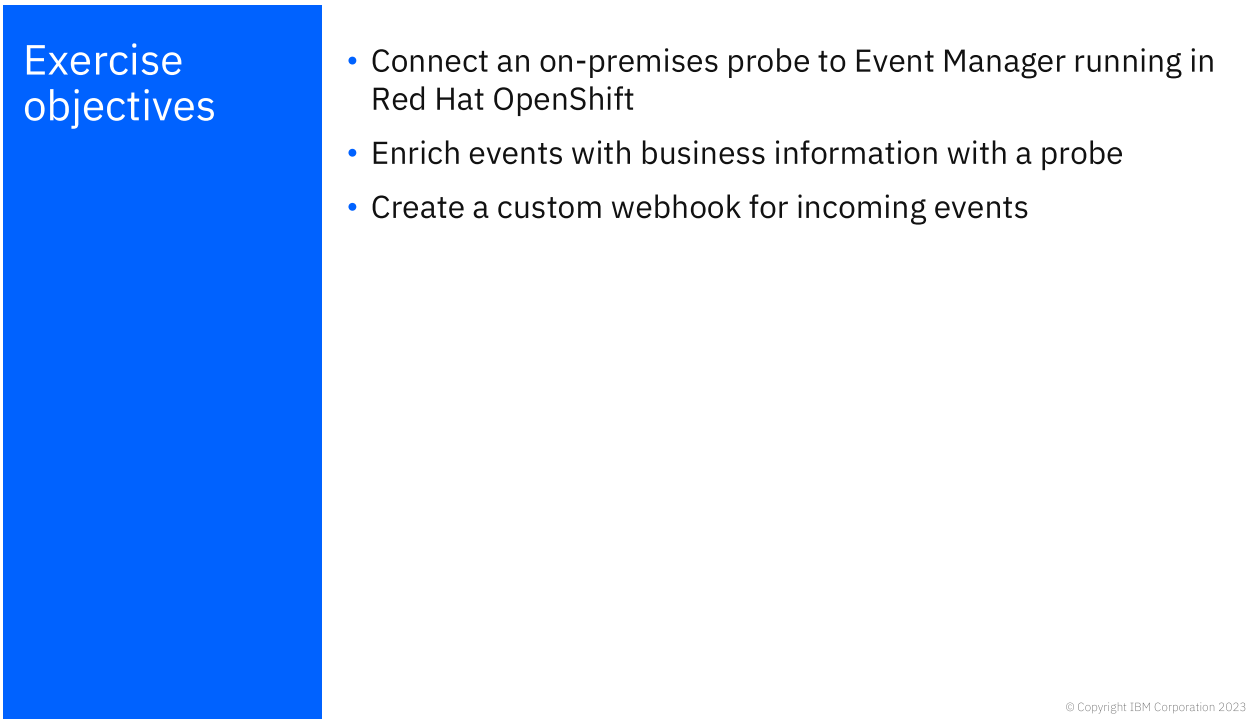


Figure 2-23. Exercise objectives

These are the tasks you complete during the lab exercises for this unit.

Lab tips

- You edit several text files in these labs. Take your time and make sure your changes are accurate before you save and close each file.
- Don't forget to copy your webhook URL. After you save your webhook, you will not be able to see the URL again.

Figure 2-24. Lab tips