



Cisco Crosswork Workflow Manager 2.0 Administrator Guide

First Published: 2025-04-25

Americas Headquarters

Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
USA
<http://www.cisco.com>
Tel: 408 526-4000
800 553-NETS (6387)
Fax: 408 527-0883



CONTENTS

CHAPTER 1	Install CWM using Docker Installer Tool 1
	Install CWM using Docker Installer Tool 1
	Prerequisites 1
	Use script to Deploy Crosswork and CWM 1
CHAPTER 2	System 5
	Architecture overview 5
CHAPTER 3	API 7
	CWM API Overview 7
	Use the CNC Workflow Automation Postman collection 7
CHAPTER 4	Events 9
	Event handling overview 9
	Brokers and protocols 9
	Kafka broker 9
	AMQP protocol (such as the RabbitMQ broker) 10
	HTTP protocol 10
	Event system configuration 11
	Event system configuration: secrets 12
	Event system configuration: resources 12
	Event type 14
	Correlation attributes 14
	Event message format 15
	Workflow event definition and state 15
	Define a Kafka event 16

Step 1: Create a Kafka secret	16
Step 2: Create a Kafka resource	16
Step 3: Add the event type	17
Step 4: Define the event in a workflow	18



CHAPTER 1

Install CWM using Docker Installer Tool

This section contains the following topics:

- [Install CWM using Docker Installer Tool, on page 1](#)

Install CWM using Docker Installer Tool

The CWM 2.0 is installed on the Cisco Crosswork platform by first deploying the Crosswork OVA file using a Docker image on the VMware vCenter 7.0 (or higher) and then installing the CWM CAPP file using the installation script.

Prerequisites

- **VMware vCenter Server 7.0** (U3p or later) and **ESXi 7.0** (U3p or later). Refer to the [Crosswork Network Controller 7.0 installation requirements](#) for more details.
- **Docker** version 19 or higher.
- `sshpass` installed. For Mac, you can use `brew install sshpass`.

Use script to Deploy Crosswork and CWM

Procedure

Step 1 In your Docker-capable machine, create a directory where you will store all the files you will use during this installation.

Note

If you are using a Mac, ensure that the directory name is in lower case.

Step 2 Download the OVA file containing the Crosswork Network Controller package from [cisco.com](#) to the directory you created. It will contain the Crosswork tar.gz CAPP file, the CWM .ova file, the `install.sh` installation script, the `configuration.json` file and Docker installer image tar.gz (along with this instruction).

Step 3 Import the Docker installer image by running the following command. Be sure to adjust the image name as needed:

```
docker image import <docker-image-name>.tar.gz your-image-name:your-tag
```

Step 4 Inside the directory, create a .txt file and paste the VMware installation template given below. For this instruction, we'll name the file `deployment.tfvars.txt` for example purposes.

```
Cw_VM_Image = ""      # Line added automatically by installer.
ClusterIPStack      = "IPv4"
DataIPNetmask        = "255.255.255.0"
DataIPGateway        = "192.168.1.1"
DNS                  = "DNS"
DomainName           = "domain_name"
CWPassword            = "your_crosswork_password"
VMSize               = "XLarge"
vm_sizes = {
  "xlarge" = {
    vcpus      = 24
    cpu_reservation = 24000
    //Memory in Mbytes
    memory = 128000
  }
}
NTP                  = "ntp.esl.cisco.com"
Timezone             = "Europe/Paris"
EnableSkipAutoInstallFeature = "True"
ManagementVIP        = "your_mgmt_vip"
ManagementIPNetmask  = "255.255.255.0"
ManagementIPGateway  = "your_mgmt_gateway"
ThinProvisioned      = "true"
DataVIP              = "your_data_vip"
CwVMs = {
  "0" = {
    VMName           = "your_VM_name",
    ManagementIPAddress = "your_mgmt_ip",
    DataIPAddress     = "your_data_ip",
    NodeType          = "Hybrid"
  }
}
VCenterDC = {
VCenterAddress = "your_vcenter_address",
VCenterUser    = "your_username",
VCenterPassword = "your_password",
DCname         = "your_datacenter_name",
MgmtNetworkName = "VM Network",
DataNetworkName = "SVM Data Network"
VMs = [{
  HostedCwVMs = ["0"],
  Host = "your_VM_host",
  Datastore = "your_VM_datastore",
  HSDatastore = "your_VM_hsdastore"
}]
}
SchemaVersion = "7.1.0"
```

Note

Note the difference between your VCenter and Datacenter.

Step 5 Edit the parameters to match your deployment.

Note

To learn more about the installation parameters, please refer to the [Single VM chapter in the Cisco Crosswork Network Controller 7.0 Installation Guide](#).

Step 6 Inside the directory, create another file named `product.json` file and paste the data below.

```
{
  "product_id": "CWM",
  "attribute": {
    "key1": "value1",
    "key2": "value2"
  }
}
```

Step 7 Open the `configuration.json` file and provide the following parameters to match your deployment:

```
{
  "SVM_NAME": "your_VM_name",
  "host": {
    "remote_user": "your_username",
    "remote_password": "your_password",
    "remote_host": "your_scp_host",
    "remote_port": "22",
    "capp_file": "/path/to/capp_file.tar.gz"
  },

  "cwm_login": {
    "ip": "your_mgmt_ip",
    "cwm_user": "admin",
    "cwm_old_password": "admin",
    "cwm_password": "your_new_password"
  },

  "deployment": {
    "tfvars_path": "/path/to/deployment.tfvars.txt",
    "ova_file": "/path/to/cwm.ova",
    "product_json": "/path/to/product.json"
  }
}
```

- for `host`, provide the details of the SCP server where your Crosswork CAPP file is located like host address and port, your username and password, and the path to the file.
- for `cwm_login`, provide your management IP and the default Crosswork username and password. In `cwm_password`, provide the new password to replace the default one upon installation completion.
- for `deployment`, provide the local paths to the `deployment.tfvars.txt` created in a previous step, to the CWM OVA file and to the `product.json` file.

Step 8 From the directory, run the installer script:

```
bash install.sh
```

This will start the installation process for the Crosswork platform and then for CWM once the platform is deployed.

Step 9 To follow the installation inside the Docker container, run the following command:

```
sudo docker ps -a
```

Copy the ID of the container in which the installation started. Usually its name contains the OVA filename, such as:
`cw-na-cwm-7.1.0-20-release-cnc710-250512-cwm-59-50`

To see the logs, run:

```
sudo docker logs your_container_id -f
```

Step 10 Once the installation script is done and the deployment status reaches 100%, go to `http://your_mgmt_vip_address:30603` and log in with the default `admin` user and the password you provided in `configuration.json`.



CHAPTER 2

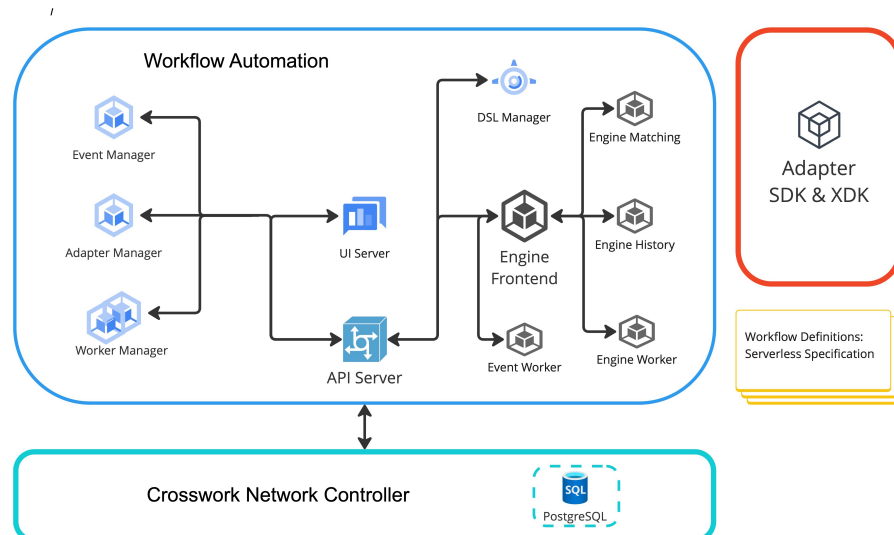
System

This section covers the following topics:

- [Architecture overview, on page 5](#)

Architecture overview

Cisco Crosswork Workflow Manager 2.0 architecture is a microservice-based solution that operates on top of the CNC platform. This section shows a diagram presenting its core architectural components along with short descriptions of each.



© 2025, Cisco Inc. or its affiliates. All rights reserved.

- **UI Server:** Allows operators to add and instantiate workflows, enter workflow data, list running workflows, monitor job progress. The **Administration** section of the CNC UI enables users to add workers, manage worker processes and assign activities from adapters to workers.
- **REST API:** Includes all interaction with the CWM application: deploying adapters, publishing and instantiating workflows, managing workers, resources and secrets.
- **API Server:** Dispatches API requests to relevant microservices.

- **Engine:** The core component that conducts how workflows are handled. It interprets and manages the execution of workflow definitions.
- **Engine Worker (Workflow Worker):** Executes the workflow tasks. It receives the workflow tasks from the **Engine**, executes them in the correct order, and sends the results back to the **Engine**.
- **Worker Manager:** Manages the Workflow Workers. It ensures that the correct number of workers are running and that they are properly configured.
- **Adapter Manager:** Manages the adapters used by the system. It installs, configures, and updates adapters ("plugins") and ensures that they are compatible with the system.
- **Event Manager:** Manages incoming and outgoing events, dispatching them to correct event queues. Events are signals coming from external sources with which the workflows can interact.
- **Adapter SDK & XDK:** Helps developers create new adapters to integrate with external systems. The XDK application extends the capabilities of the Adapter SDK to enable developers to automatically build interfaces and message logic for custom adapters.
- **Workflow Definitions:** Workflow code written in the JSON format based on the Serverless Workflow specification.
- **Crosswork Network Controller (CNC):** Runtime platform for the CWM application. It is a collection of services that provide the necessary infrastructure to support the deployment and management of the application within a Cluster deployment.
- **PostgreSQL:** The database that the system uses to store and manage its data.
- **DSL Engine:** Executes the Domain-Specific Language (DSL) used to define the workflows. It parses the DSL, generates the appropriate workflow code, and compiles it for execution.
- **Engine Matching:** Matches incoming events with the appropriate workflow. It determines which workflow should execute based on the event data and the defined workflow constraints.
- **Engine History** Tracks the history of executed workflows. It stores the metadata and execution details of all completed, running, and failed workflows.



CHAPTER 3

API

This section covers the following topics:

- [CWM API Overview, on page 7](#)
- [Use the CNC Workflow Automation Postman collection, on page 7](#)

CWM API Overview

Cisco developed the Cisco Crosswork Workflow Manager API based on Representational State Transfer (REST) design principles. You can access the API using HTTP and data files formatted using JSON. The API indicates the success or failure of a given request using relevant HTTP response codes. Data retrieval methods require a GET request, while methods for adding, changing, or deleting data require POST, PUT, PATCH, or DELETE methods, as appropriate. The API returns errors if you send requests using the wrong request type.

You can use the CWM API using a **CWM 2.0 Postman collection** in Postman.

For a full API reference, see the dedicated DevNet space: <https://devnetapps.cisco.com/docs/crosswork/workflow-manager/introduction/>.

Use the CNC Workflow Automation Postman collection

Follow these steps to import the collection to the Postman application and set the development environment.

Before you begin

Be sure that you have access to a Postman web application account or have installed the Postman desktop app. For details, see <https://www.postman.com/downloads/>.

You must also download the CNC Workflow Automation [Postman collection in JSON format](#) by clicking [this link](#) and then unzip the archive to an accessible storage resource.

Procedure

Step 1 Launch Postman and go to **Collections**.

- Step 2** Click **Import**, select **folders** from the **Drop anywhere to import** screen, and point to the folder that you unzipped from the CNC Workflow Automation Postman collection archive.
- Step 3** Go to **Environments** and select the newly imported **test** environment.
- Step 4** Provide current values for the **baseUrl** and **endPoint** variables to match the IP address and port of your CNC Workflow Automation instance. Save the changes.

To get access to the CNC Workflow Automation API, use the `baseUrl/crosswork/cnc/v71/`, where `baseUrl` is the IP address and port number of your Crosswork Network Controller (CNC) instance with CNC Workflow Automation installed. For example: `https://172.22.141.178:30603`



CHAPTER 4

Events

This section covers the following topics:

- [Event handling overview, on page 9](#)
- [Define a Kafka event, on page 16](#)

Event handling overview

The event handling mechanism enables CWM to interact with external brokers for handling outbound and inbound events. Workflows can act as either consumers or producers of events which can be used to initiate a new workflow, or signal an existing workflow. For each event type that you define, you can add correlation attributes for filtering events and routing them to the workflow waiting for the event containing specific attribute values.

Event messages need to be defined according to [Cloud Events specification](#). See [Event message format, on page 15](#) for more details.

Brokers and protocols

CWM supports the Kafka broker and the AMQP and HTTP protocols for handling events. Events can be either **consumed** by a workflow running inside CWM (incoming events forwarded by a broker) or **produced** by a running workflow and forwarded to an external system (outgoing events received by a broker).



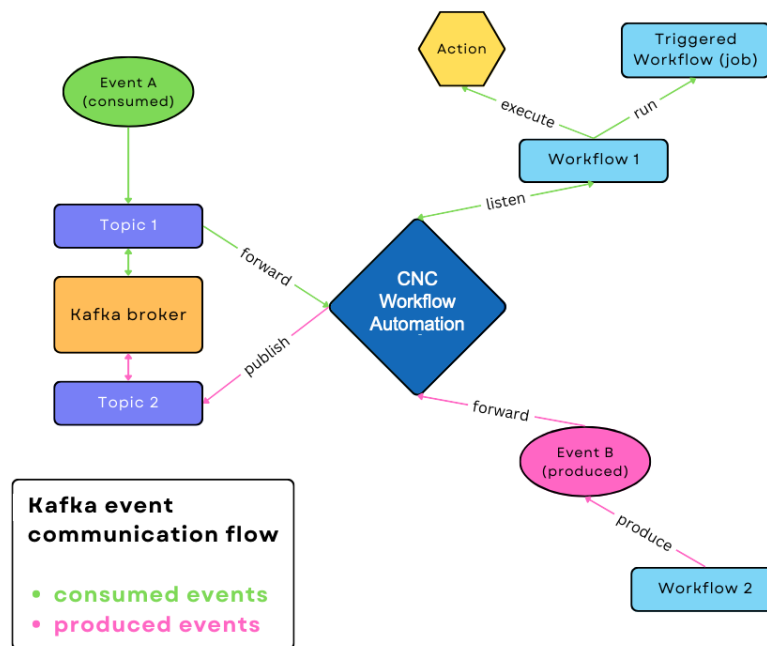
Note

It is important to remember that CWM doesn't act as an event broker itself. It provides a means to connect to external brokers to forward messages and events.

Kafka broker

For the **consume** event type, CWM connects to a Kafka broker and listens for a specific event type on a topic. Once an event of the specific type registers to the right topic, CWM retrieves the event data and forwards it to the running workflow. The workflow then executes actions defined inside the Event State and/or runs another workflow execution (if selected).

For the **produce** event type, a running workflow produces a single event or a set of events which CWM then forwards to the broker and they get published in the right topic.



The Kafka broker will accept every event message format supported by the language-specific SDK as long as a valid content-type is sent. See this Github link for lists of supported formats: <https://github.com/cloudevents/spec?tab=readme-ov-file>.

AMQP protocol (such as the RabbitMQ broker)

For the **consume** event type, CWM connects to an AMQP broker and listens for a specific event type on a queue. Similarly to the Kafka broker, when an event of the specific type registers to the right queue, CWM retrieves the event data and forwards it to the running workflow. The workflow then executes actions defined inside the Event State and/or runs another workflow execution (if selected).

For the **produce** event type, a running workflow produces a single event or a set of events which CWM then forwards to the broker and they get published in the right queue.

AMQP brokers will accept every event message format supported by the specific SDK as long as a valid content-type is sent. The lists of supported event formats are available here: <https://github.com/cloudevents/spec?tab=readme-ov-file>.

HTTP protocol

For the **consume** event type, CWM exposes an HTTP endpoint that listens for any incoming events. If an event of a specific type comes, it is forwarded to the running workflow that waits for this event type.



Note When events are consumed, CWM functions as the destination HTTP server. Therefore, the URL of the CWM server is what you effectively provide as the resource for the given HTTP event type.

Event messages need to be HTTP *POST* requests, and the message body needs to be in JSON format representing a Cloud Event:

```
{ "specversion": "1.0",
  "id": "2763482-4-324-32-4",
  "type": "com.github.pull_request.opened",
  "source": "/sensors/tn-1234567/alerts",
  "datacontenttype": "text/xml",
  "data": "<test=\"xml\"/>",
  "contextAttrName": "contextAttrValue" }
```

For **produce** events, a workflow produces an event in the Cloud Event format and CWM forwards it as an HTTP *POST* request to an HTTP endpoint exposed by an external system. The HTTP endpoint address is a concatenation of the host **URL** defined in the Resource configuration in CWM and the **End point** field of the Event definition inside the workflow definition. Inside the resource configuration, you can change the request method to *PUT* or other, and add key and value pairs as header (in JSON format):

Administration > Resources

New resource

Cancel Create

General

Resource name*

Resource type *

Secret ID

Connection

url*

method

headers

```
{
  "key1": "value1",
  "key2": "value2"
}
```

Event system configuration

The following topics cover the details of event configuration.

Event system configuration: secrets

In event configuration, secrets store credentials needed to enable connection to a broker or endpoint exposed by a third-party service that sends or receives events. This includes basic authentication: username and password. The Secret ID that you provide when creating a secret will be referenced when creating a resource, so you need to add a secret beforehand. For details, see [Step 1: Create a Kafka secret, on page 16](#).

Event system configuration: resources

The resource is where you provide all the connection details (including the secret) needed to reach an event broker or endpoint exposed by a third party service. Depending on the broker/protocol you want to use, you can choose among three default event resource types

- `system.event.amqp.v1.0.0`
- `system.event.kafka.v1.0.0`
- `system.event.http.v1.0.0`

Notice that there is a different set of configuration fields for each of them

- For AMQP, provide the **ServerDSN** in the following format `amqp //localhost 5723`.
- For Kafka:
 - **KafkaVersion**: Enter your Kafka version. The standard way to check the Kafka version is to run `bin/kafka topics.sh version` in a terminal.
 - **Brokers**: Enter your Kafka broker addresses in the following format `["localhost 9092", "192.168.10.9 9092"]`.
 - **OtherSettings**: An editable list with default Kafka setting values. You can modify the values as needed. For details, see the "Kafka Other Settings" table below.
- For HTTP:
 - **Produce** event types: Fill in the **URL** field and optionally, **Method** and **Headers** (for example, Client ID header name and value as a JSON object).



Note

The **URL** needs to be the address of the destination HTTP server, but without the URL path. You will enter the URL path as the **End point** when configuring the event type.

- **Consume** event types: Fill in the **URL** field with the server URL of your CWM instance, for example, `192.168.10.9 9092`.



Note

Remember to provide the URL of your CWM instance without the URL path (`/event/http`). You will enter the URL path as the **End point** when configuring the event type.

Table 1: Kafka Other Settings

Field	Description
ClientID	The identifier used by Kafka brokers to track the source of requests
KafkaVersion	Specifies the version of Kafka the client is compatible with (e.g., "2.0.0")
MetadataFull	When True, fetches metadata for all topics, not just those needed
AdminRetryMax	Maximum number of retries for admin requests (e.g., creating/deleting topics)
NetSASLVersion	Version of the SASL (Simple Authentication and Security Layer) protocol
AdminTimeoutSecs	Timeout in seconds for admin requests (e.g., topic creation)
ConsumerFetchMin	Minimum amount of data in bytes the broker should return to the consumer
MetadataRetryMax	Maximum number of retries to fetch metadata (e.g., topic and partition info)
NetSASLHandshake	When True, enables the SASL handshake mechanism
NetDialTimeoutSecs	Timeout in seconds for establishing a connection to Kafka
NetReadTimeoutSecs	Timeout in seconds for reading data from Kafka
NetWriteTimeoutSecs	Timeout in seconds for writing data to Kafka
ProducerTimeoutSecs	Timeout in seconds for producing messages to Kafka
ConsumerFetchDefault	Default size in bytes for the consumer fetch request (e.g., 1MB)
ProducerRequiredAcks	Specifies the required number of acknowledgments from brokers for a message to be considered successful (e.g., "WaitForLocal")
ProducerReturnErrors	When True, enables error reporting for failed produce requests
ConsumerIsolationLevel	Specifies whether the consumer reads uncommitted or committed messages ("ReadUncommitted" allows reading in-progress transactions)
ConsumerOffsetsInitial	Initial offset when there is no committed offset (-1 for the latest)

Field	Description
NetMaxOpenRequestsSecs	Maximum time for open requests over the network

Event type

To create a new event type, you need to have a resource and secret added to CWM.

The following fields are available when adding an event type:

- **Event type name:** the name of your event type. It's later referred to inside the workflow definition.
- **Resource:** a list of resources previously added to CWM.
 - **Event source:** a fully user-defined entry that will be referenced in the workflow definition. Required for `produce` event kind.
 - **End point:** the name of Kafka topic (event stream), AMQP endpoint (terminus), or HTTP URL (Host) path.



Note For the HTTP **consume** event type, provide `/event/http` as your **End point**.

- **Select kind:** a list consisting of two options: `consume` or `produce` event kind.



Note The `both` option is not yet supported for CWM.

- **Start listener** (only for `consume` kind): check it to start listening for the defined event type.
- **Run job** (only for `consume` kind): tick this checkbox if you want to trigger a workflow upon receiving the event. Then select the desired workflow from the list.

Correlation attributes

Optionally, you can set context attributes for your event. They apply only to the `consume` event kind and are used to trigger workflows selectively. You can view them as a kind of custom filters that refine the inbound event data and route them to the right workflows that listen on event types with specific values of correlation attributes.

To add an attribute to your event type, click **Add attribute**, and provide an attribute name.



Note Correlation attributes are fully user-defined. They need to match the JSON key and value pair stated inside the Cloud event message that is to be routed to a given workflow.

Event message format

Event messages must follow the [Cloud Events specification](#) format. A minimum viable event message following the specification will contain the following parameters:

```
{
  "specversion": "1.0",
  "id": "00001",
  "type": "com.github.pull_request.opened",
  "source": "/sensors/tn-1234567/alerts"
}
```

The message can carry additional parameters, such as "datacontenttype", "data", and a correlation context attribute name (contextAttrName in this example) :

```
{
  "specversion": "1.0",
  "id": "2763482-4-324-32-4",
  "type": "com.github.pull_request.opened",
  "source": "/sensors/tn-1234567/alerts",
  "datacontenttype": "text/xml",
  "data": "<test data=\"xml\"/>",
  "contextAttrName": "contextAttrValue"
}
```

Workflow event definition and state

In the workflow definition, there are two major syntactical elements that you use to handle the events for which the workflow will be waiting. These are:

- The [Event definition](#): Used to define the event type and its properties. For example:

```
{
  "name": "applicant-info",
  "type": "org.application.info",
  "source": "applicationssource",
  "correlation": [
    {
      "contextAttrName": "applicantId"
    }
  ]
}
```

- The [Event state](#): Used to define actions to be taken when the event occurs. For example:

```
{
  "name": "MonitorVitals",
  "type": "event",
  "onEvents": [
    {
      "actions": [
        {
          "functionRef": {
            "refName": "uppercase",
            "arguments": {
              "input": {
                "in": "patient ${ .patient } has high temperature"
              }
            }
          }
        }
      ]
    }
  ]
}
```

```

    ],
    "eventRefs": [
      "HighBodyTemperature"
    ]
  }
]
}

```

Define a Kafka event

In the following topics, we will create a Kafka event and add it to a new workflow. The only pre-requisites are that we must have:

- A fully set-up Kafka service.
- CWM installed.

Step 1: Create a Kafka secret

To enable a secure connection to the Kafka service, you need to create a secret with Kafka credentials and a resource with connection details.

Procedure

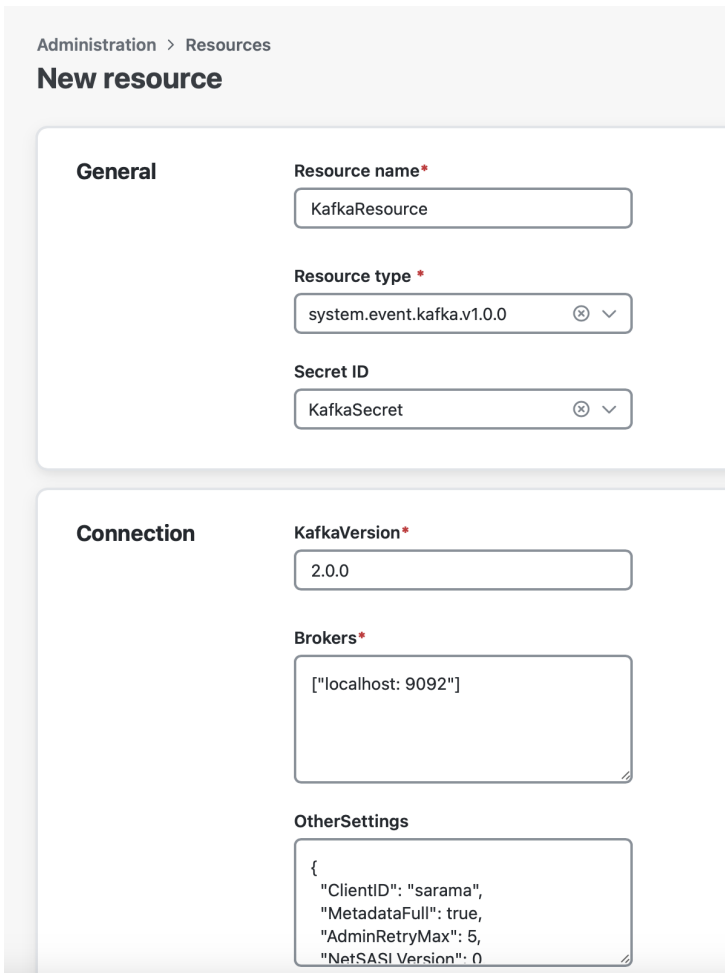
	Command or Action	Purpose
Step 1	In CNC, select Administration > Workflow Administration > Secrets .	
Step 2	Click Add Secret .	
Step 3	In the New secret view, specify the following:	
Step 4	After selecting the secret type, a set of additional fields is displayed under the Secret type details section. Fill in the fields:	
Step 5	Click Create Secret .	

Step 2: Create a Kafka resource

You also need to create a resource with connection details.

Procedure

	Command or Action	Purpose
Step 1	In CNC, select Administration > Workflow Administration > Resources .	
Step 2	Click Add Resource .	

	Command or Action	Purpose
Step 3	In the New resource window, specify the following:	
Step 4	Click Create .	

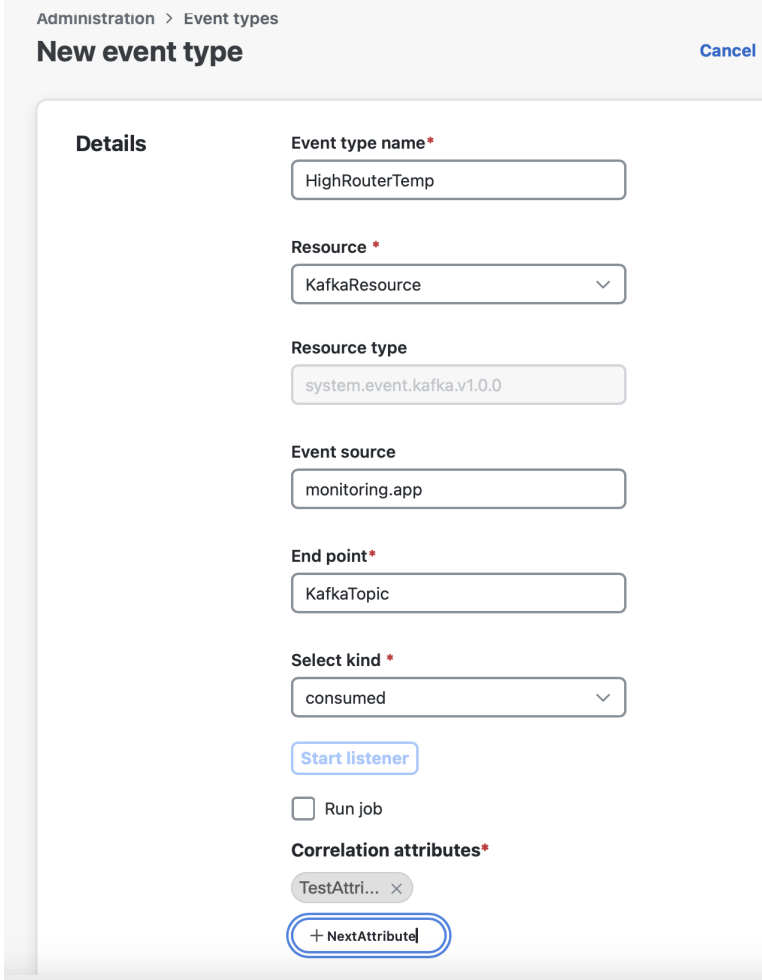
Step 3: Add the event type

When you have the secret and resource in place, it's time to specify the type of event that will be consumed or produced.

Procedure

	Command or Action	Purpose
Step 1	In CNC, select Administration > Workflow Administration > Event Types .	
Step 2	Click Add event type .	
Step 3	In the New event type window, provide the required input:	

Step 4: Define the event in a workflow

	Command or Action	Purpose
Step 4	Click Create Event type .	

Step 4: Define the event in a workflow

Now that we have the event type added, we can create a workflow that registers for this event type and executes an action when the event is received by CWM. To do so, we'll need to:

1. Define the event using an [Event definition](#).
2. Specify the [Event state](#)
3. Define the actions to be taken when the event occurs.

As an example, let's take a scenario where a router overheating alarm (an inbound event) triggers a single workflow event state and defines two remediation actions to be executed in response to that state.

```
{
  "id": "HighRouterTempWorkflow",
  "name": "Router Overheating Alarm Workflow",
  "start": "RemediateHighTemp",
  "events": [
```

```

    {
      "kind": "consumed",
      "name": "HighRouterTemp",
      "type": "HighRouterTemp",
      "source": "monitoring.app"
    }
  ],
  "states": [
    {
      "end": {
        "terminate": true
      },
      "name": "RemediateHighTemp",
      "type": "event",
      "onEvents": [
        {
          "actions": [
            {
              "functionRef": {
                "refName": "DispatchTech",
                "contextAttributes": {
                  "RouterIP": "${ .RouterIP }"
                },
                "resultEventTimeout": "PT30M"
              }
            }
          ],
          "eventRefs": [
            "HighRouterTemp"
          ]
        },
        {
          "actions": [
            {
              "functionRef": {
                "refName": "MoveTraffic",
                "contextAttributes": {
                  "RouterIP": "${ .RouterIP }"
                },
                "resultEventTimeout": "PT30M"
              }
            }
          ],
          "timeouts": {
            "actionExecTimeout": "PT60M"
          }
        }
      ]
    }
  ],
  "version": "1.0.0",
  "description": "Remediate router overheating",
  "specVersion": "0.8"
}

```

**Note**

This example is not a complete workflow. It is an example of how to define an event inside a workflow, set a simple state, and then define actions to take in response to that single state. A realistic workflow can define many more states and actions to take in response to each of those states.

