



Cisco Crosswork Workflow Manager 1.1 Administrator Guide

First Published: 2024-03-25

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



CHAPTER 1

Install CWM using OVA

This section contains the following topics:

- [Install CWM using OVA, on page 1](#)

Install CWM using OVA

The Crosswork Workflow Manager 1.1 is installed as a guest virtual machine by deploying an OVA image using the VMware vSphere 6.7 (and higher) virtualization platform.

Prerequisites

- An **ed25519** SSH public and private key pair.

System requirements

Minimum system requirements	
Server	VMware vSphere 6.7+ account with an ESXi 6.7+ host
CPU	8 cores
Memory	64 GB
Storage	100 GB

Download the CWM package

To get the CWM 1.1 software package:

- Step 1** Go to the Cisco Software Download service and in the search bar, type in '**Crosswork Workflow Manager**', then select it from the search list.
- Step 2** From Select a software type, select **Crosswork Workflow Manager Software**.
- Step 3** Download the Crosswork Workflow Manager software package for Linux.

- Step 4** In a terminal, use the `sh` command to extract the downloaded **.signed.bin** file and verify the certificate. See example output below for reference:

```
sh cwm-1.1.signed.bin
Unpacking...
Verifying signature...
Retrieving CA certificate from http://www.cisco.com/security/pki/certs/crcam2.cer ...
Successfully retrieved and verified crcam2.cer.
Retrieving SubCA certificate from http://www.cisco.com/security/pki/certs/innerspace.cer ...
Successfully retrieved and verified innerspace.cer.
Successfully verified root, subca and end-entity certificate chain.
Successfully fetched a public key from tailf.cer.
Successfully verified the signature of cwm-1.1.tar.gz using tailf.cer
```

The `cwm-1.1.tar.gz` file and other files have been extracted and validated against the signature file.

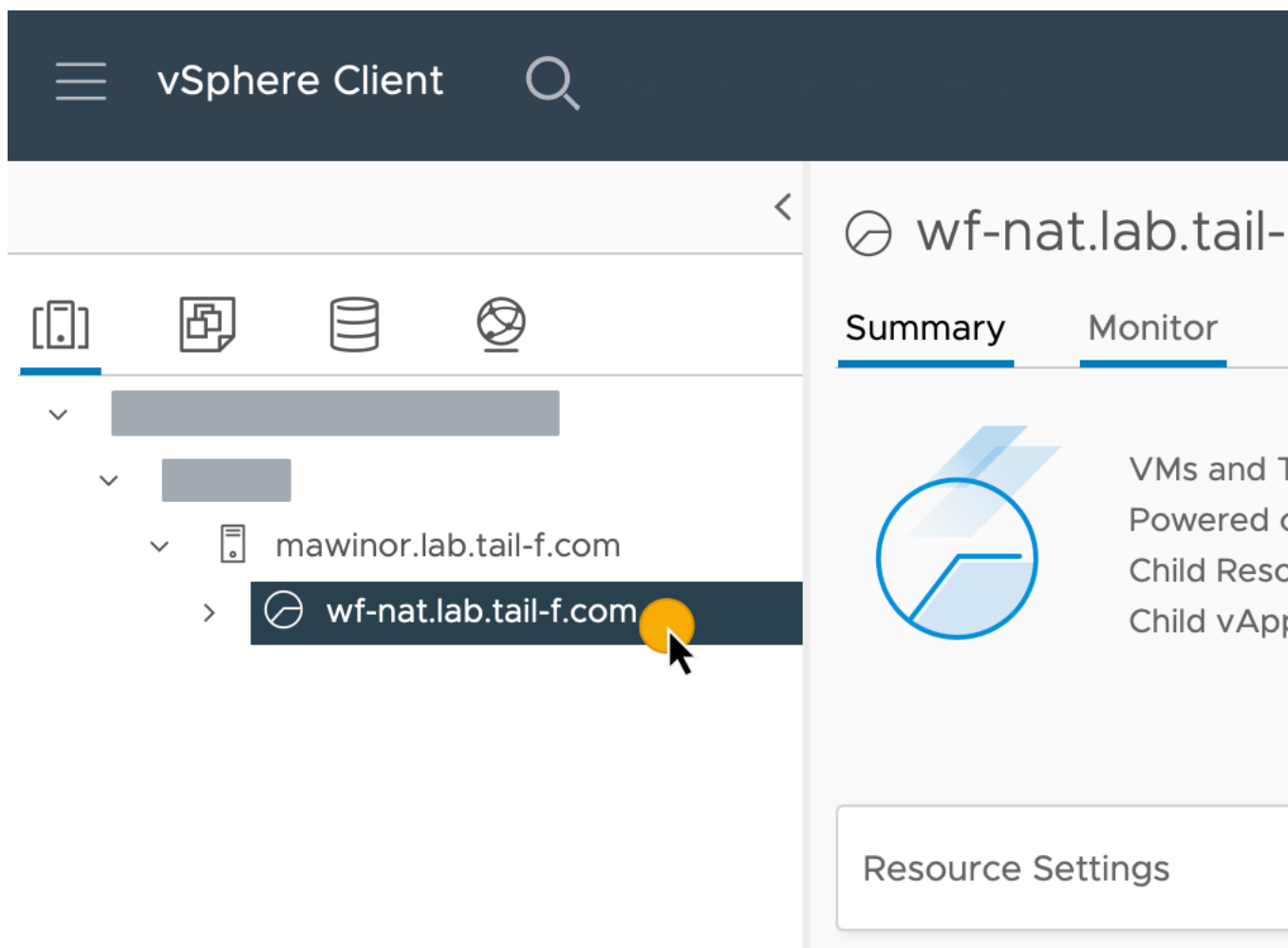
- Step 5** To extract the `cwm-1.1.tar.gz` file, double click on it (Mac users) or use `gzip` utility (Linux and Windows users). This will extract the CWM OVA file that will be used for installation.
-

Deploy OVA and start VM

To create a virtual machine using the downloaded OVA image:

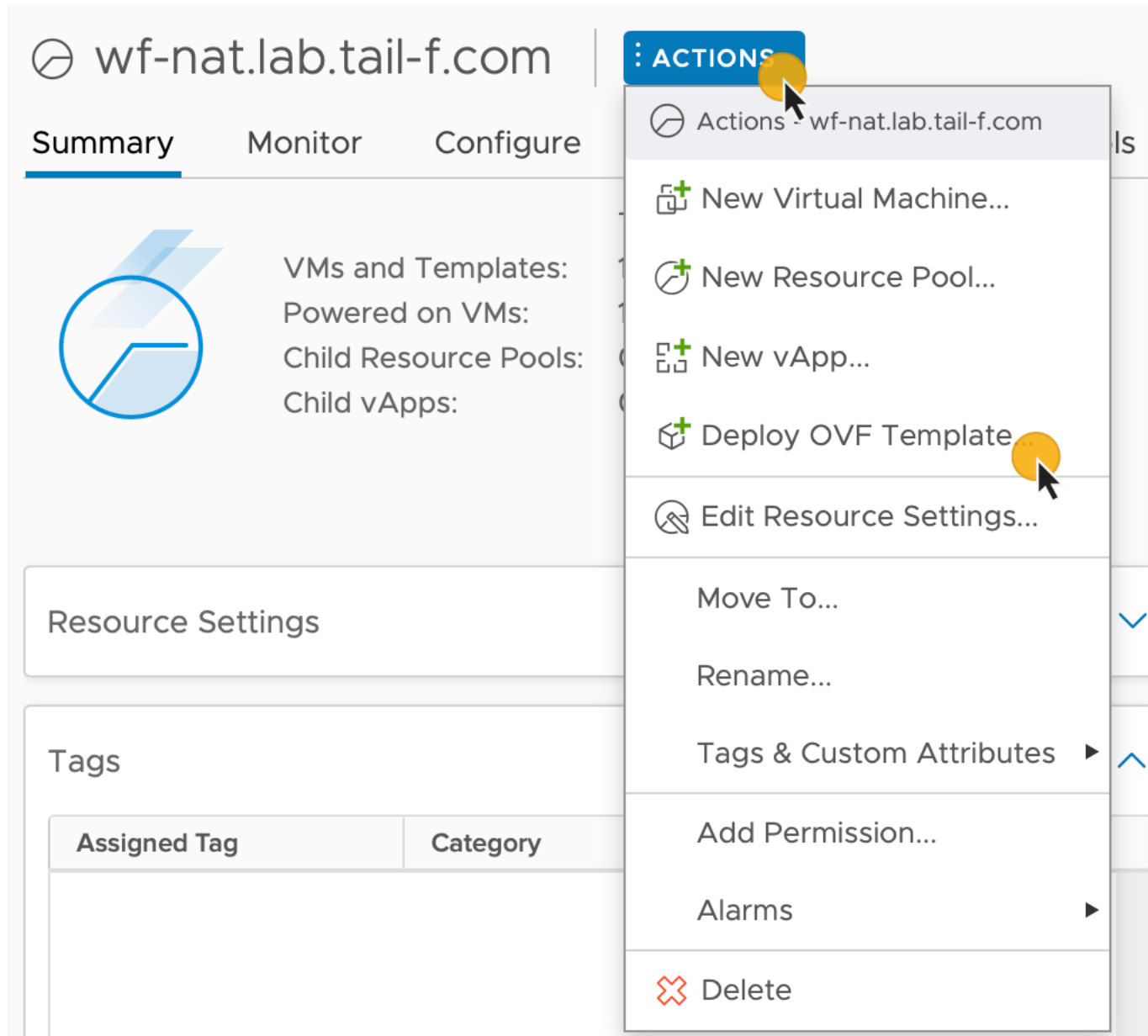
- Step 1** Log in to your vSphere account.
- Step 2** In the **Hosts and Clusters** tab, expand your host and select your resource pool.

Figure 1:



Step 3 Click the **Actions** menu and select **Deploy OVF Template**.

Figure 2:



- Step 4** In the **Select an OVF template** step, click **Local file**, **Select files**, and select the CWM OVA image. Click **Next**.
- Step 5** In the **Select a name and folder** step, provide a name for your VM and select its location. Click **Next**.
- Step 6** In the **Select a compute resource** step, select your resource pool. Click **Next**.
- Step 7** In the **Review details** step, click **Next**.
- Step 8** In the **Select storage** step, set **Select virtual disk format** to **Thin provision** and select your storage, then click **Next**.
- Step 9** In the **Select network** step, you need to select destination networks for the **Control Plane** and **Northbound**:
- a) **Control Plane**: select **PrivateNetwork**. If not available, select **VM Network**.

Note Control plane settings are essential only in case of an HA cluster setup. For single-node setups, control plane settings need to be provided, but are not essential and should not conflict with any other devices connected to the control network.

- b) **Northbound:** select **VM Network**.
- c) Click **Next**.

Step 10 In the **Customize template** step, provide the following selected properties:

- a) **Instance Hostname:** type a name for your instance.
- b) **SSH Public Key:** provide an **ed25519** SSH public key that will be used for command-line access to the VM.
- c) **Node Name:** provide a name for installation node.

Note For single-node setups, it's not recommended to modify the node name. If you modify it, remember that it must match the **Zone-A Node Name** below.

- d) **Control Plane Node Count:** change to more than 1 only in case of HA cluster setup. Not supported for CWM 1.1.
- e) **Control Plane IP (ip subnet):** provide a network address for the control plane. This address cannot conflict with any other devices in the control network, but is otherwise inessential in a single-node setup. Note that the default subnet mask is /24. You can add your custom subnet mask value if applicable for your network settings.
- f) **Initiator IP:** set the initiator IP for the starter node. In a single-node setup, it is the same address as *Control Plane IP**.

Deploy OVF Template

- 1 Select an OVF template
- 2 Select a name and folder
- 3 Select a compute resource
- 4 Review details
- 5 Select storage
- 6 Select networks
- 7 Customize template**
- 8 Ready to complete

Customize template

Customize the deployment

General	Instance Hostname
	SSH Public Key
Node Config	Node Name
	Data Volume Size (GB)
	Cluster Join Token
	Control Plane Node Count
	Control Plane IP (ip[0])
	Initiator IP

- Cisco Crosswork Workflow Manager 1.1 Administrator Guide

- m) **Zone-C Node Name (Arbitrator)**: provide the name of the Zone-C Arbitrator node. For single-node setups, this is not essential and must not be modified.
- n) Click **Next**.

Figure 4:

Deploy OVF Template

- 1 Select an OVF template
- 2 Select a name and folder
- 3 Select a compute resource
- 4 Review details
- 5 Select storage
- 6 Select networks
- 7 Customize template**
- 8 Ready to complete

Customize template

Northbound Interface

Protocol

IP (ip[/subnet]) - if not

Gateway - if not using

DNS

Initiator Config

Initiator Node

Northbound Virtual IP

Zone-A Node Name

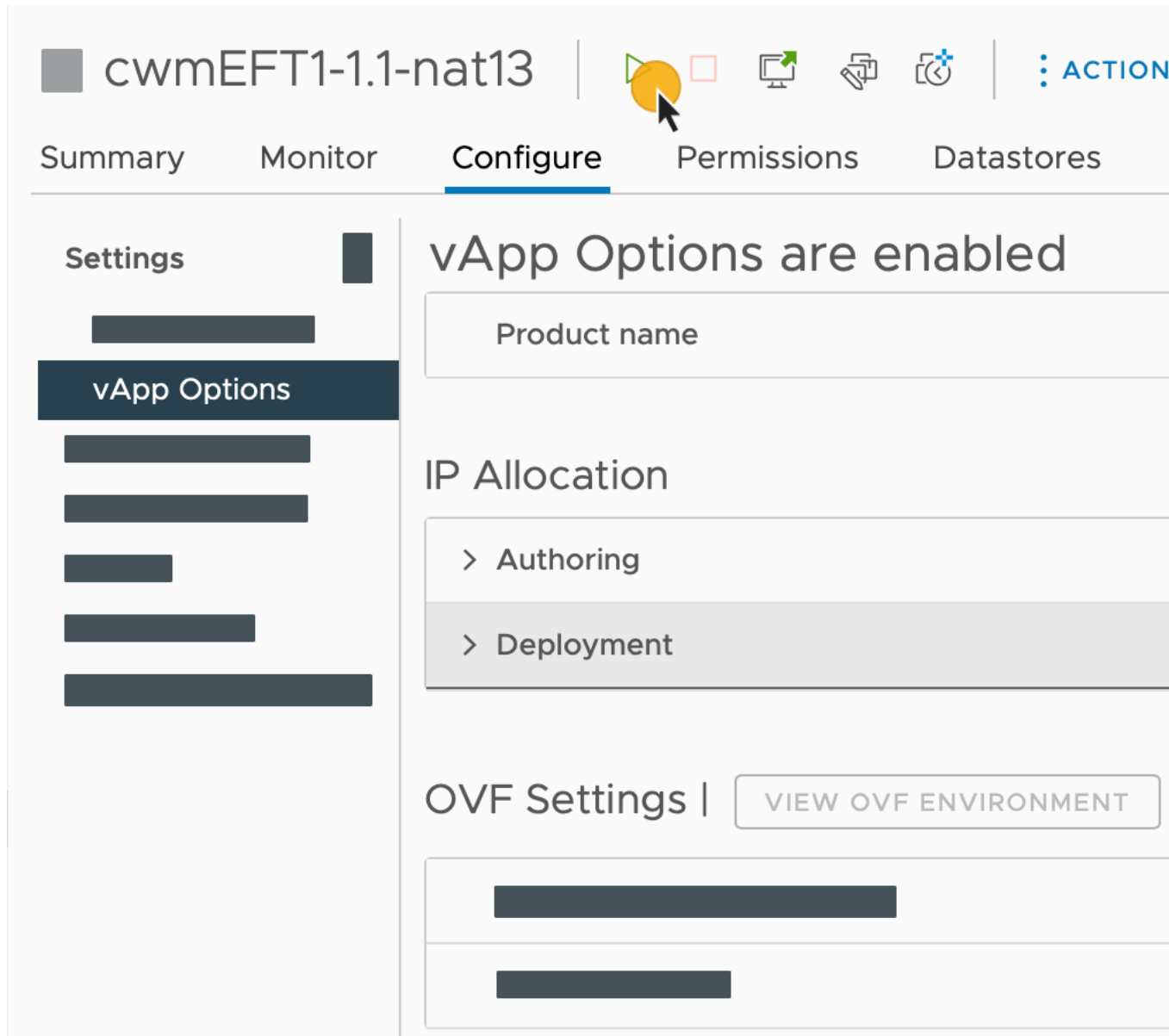
Zone-B Node Name

Zone-C Node Name

Step 11 In the **Ready to complete**, click **Finish**. The deployment may take a few minutes.

Step 12 From the **Resource pool** list, select you newly created virtual machine and click the **Power on** icon.

Figure 5:



Note If the VM doesn't power on successfully, this might be due to an intermittent infrastructure error caused by NxF. As a workaround, remove the existing VM and redeploy the OVA on a new one.

Check installation and create user

Before you create a platform user account for first login to the CWM UI, check if the installation is completed successfully and the system is up:

Step 1 Using a command-line terminal, log in to the NxF in your guest OS with SSH:

```
ssh -o UserKnownHostsFile=/dev/null -p 22 nxf@<virtual_IP_address>
```

Note By default, the virtual IP address is the one you set in **IP (ip subnet) - if not using DHCP**. Depending on how vCenter is set up, this can be the resource pool address along with a specific port. Check this with your network administrator in case of doubt

Optional: If you are logging in for the first time, provide the path name for your private key:

```
ssh -i <ed25519_ssh_private_key_name_and_location> nxf@<virtual_IP_address>
```

Note The default port for SSH is 22, change it to your custom port if applicable.

Step 2 Check NxF boot logs:

```
sudo journalctl -u nxf-boot
```

Note Note that it may take a few minutes for the installation to complete. At the bottom of the NxF logs that appear, look for the `NXF: Done setting up machine` message. If the logs report an issue, you might consider reinstalling CWM.

Step 3 Check if all the Kubernetes pods are up and running:

```
kubectl get pods -A
```

This will display a list of pods accompanied by their status, which will resemble the following:

NAMESPACE	NAME	READY	STATUS	RESTARTS
AGE				
kube-flannel 7m35s	kube-flannel-ds-vh4js	1/1	Running	0
kube-system 7m35s	coredns-9mnzv	1/1	Running	0
kube-system 7m44s	etcd-node1	1/1	Running	0
kube-system 7m50s	kube-apiserver-node1	1/1	Running	0
kube-system 7m50s	kube-controller-manager-node1	1/1	Running	0
kube-system 7m35s	kube-proxy-6hwg9	1/1	Running	0
kube-system 7m42s	kube-scheduler-node1	1/1	Running	0
local-path-storage 7m34s	local-path-provisioner-54c455f95-mbhc9	1/1	Running	0
nxf-system 6m25s	authenticator-f74c7c87f-m8p4x	2/2	Running	0
nxf-system 6m27s	controller-76686f8f5f-gpqc	2/2	Running	0
nxf-system 4m17s	ingress-ports-node1-zchwz	1/1	Running	0
nxf-system 6m23s	ingress-proxy-bcb8c9fff-lzm9p	1/1	Running	0
nxf-system 7m34s	kafka-0	1/1	Running	0
nxf-system 6m33s	loki-0	3/3	Running	0
nxf-system 6m30s	metrics-5qnzb	2/2	Running	0
nxf-system	minio-0	2/2	Running	0

7m34s					
nxf-system	postgres-0	2/2	Running	0	
6m59s					
nxf-system	promtail-t7dp4	1/1	Running	0	
6m33s					
nxf-system	registry-5486f46b54-c6tf9	2/2	Running	0	
7m2s					
nxf-system	vip-node1	1/1	Running	0	
6m12s					
zone-a	cwm-api-service-67bd9db5c7-vfszs	2/2	Running	2 (3m37s	
ago)					
4m16s					
zone-a	cwm-dsl-service-7ffd6975ff-wlrwt	2/2	Running	4 (3m21s	
ago)					
4m15s					
zone-a	cwm-engine-frontend-6754445fc-67t5h	2/2	Running	2 (3m52s	
ago)					
4m15s					
zone-a	cwm-engine-history-c4dfffd-d-t2fgv	2/2	Running	1 (2m35s	
ago)					
4m14s					
zone-a	cwm-engine-history-c4dfffd-d-wr5v2	2/2	Running	2 (3m51s	
ago)					
4m14s					
zone-a	cwm-engine-history-c4dfffd-d-zz74q	2/2	Running	4 (48s ago)	
4m14s					
zone-a	cwm-engine-matching-78dfdf858f-q8wg2	2/2	Running	2 (3m46s	
ago)					
4m14s					
zone-a	cwm-engine-ui-6b74755499-jwbld	2/2	Running	0	
4m13s					
zone-a	cwm-engine-worker-589b6bc88b-hs2ch	2/2	Running	0	
4m13s					
zone-a	cwm-event-manager-5b95bb49db-gw6g5	2/2	Running	0	
4m12s					
zone-a	cwm-plugin-manager-76f798446c-qgx27	2/2	Running	1 (2m29s	
ago)					
4m12s					
zone-a	cwm-ui-779bdb44-98d5v	2/2	Running	0	
4m11s					
zone-a	cwm-worker-manager-7bd8795b56-f4czp	2/2	Running	1 (112s ago)	
4m10s					
zone-a	logcli-5f8cc8c585-fq7wm	2/2	Running	0	
4m10s					

Note Note that it may take a few minutes for the system to get all the pods running. If any of the pods stays in a status other than Running, consider using the `kubectl delete pod <pod_name> -n <namespace>` command to restart it.

Create user for UI login

You can create CWM platform user accounts using the command-line access to the VM. Here's how to do it:

Step 1 Using a command-line terminal, log in to the NxF in your guest OS with SSH:

```
ssh -o UserKnownHostsFile=/dev/null -p 22 nxf@<virtual_IP_address>
```

Optional: If you are logging in for the first time, provide the path name for your private key:

```
ssh -i <ed25519_ssh_private_key_name_and_location> nxf@<virtual_IP_address>
```

Note The default port for SSH is 22, change it to your custom port if applicable.

Step 2 To create a user with a password, run the following commands:

a) First, set minimum password complexity (default is 3, 0 is complexity disabled):

```
sedo security password-policy set --min-complexity-score 1
```

b) Then create user account and a password:

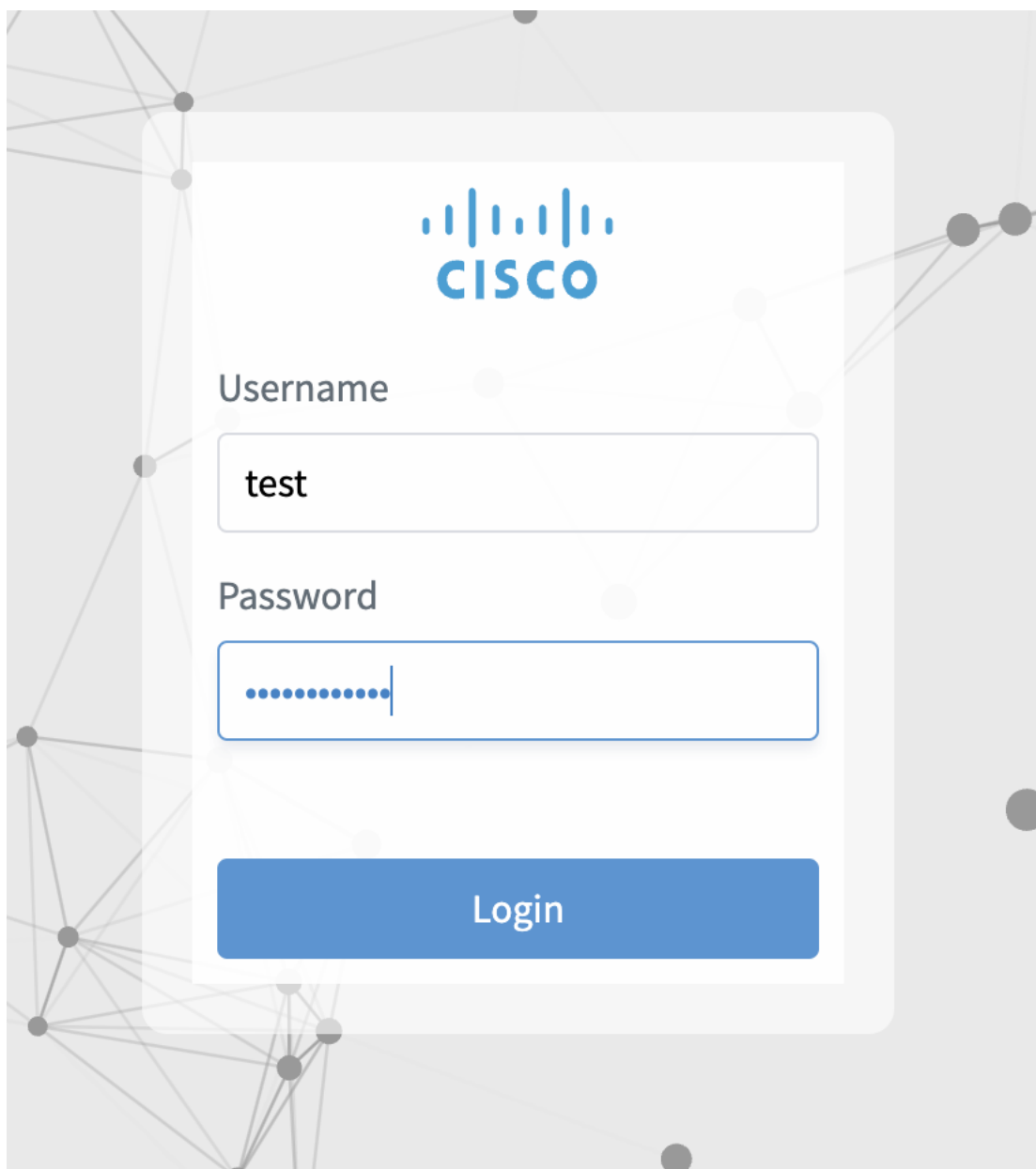
```
echo -en 'Password123!' | sedo security user add --password-stdin \  
--access permission/admin --access permission/super-admin \  
--access permission/user --display-name Tester test
```


c) Optionally, disable the password change requirement for the test user:

```
sedo security user set test --must-change-password=false
```

Step 3 To see the CWM UI, go to the address that you selected for Northbound IP and default port 8443. For example, <https://192.168.1.233:8443/>.

Step 4 Log in using the `test` username and password.

The image shows a login interface for Cisco Crosswork Workflow Manager. It features the Cisco logo at the top, followed by 'Username' and 'Password' labels. The username field contains 'test' and the password field is masked with dots. A blue 'Login' button is at the bottom. The background is a light gray with a network diagram pattern.



Username

test

Password

.....

Login



CHAPTER 2

System

This section contains the following topics:

- [Architecture overview, on page 15](#)
- [Check health and logs, on page 16](#)

Architecture overview

The Crosswork Workflow Manager architecture is a microservice-based solution that operates on top of the Kubernetes container orchestration system. This section shows a diagram presenting its core architectural components along with short descriptions of each.

- **User Interface (UI)**: allows operators to add and instantiate workflows, enter workflow data, list running workflows, monitor job progress. The **Admin** section of the UI enables adding workers, managing worker processes and assigning 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.
- **Control Server**: dispatches API requests to relevant microservices.
- **Workflow Engine**: it is the core component that conducts how workflows are handled; it interprets and manages the execution of workflow definitions.
- **Execution Engine (Workflow Worker)**: it is responsible for executing the workflow tasks. It receives the workflow tasks from the **Workflow Engine**, executes them in the correct order, and sends the results back to the **Workflow Engine**.
- **Adapter Workers**: they are processes responsible for executing the tasks defined in workflow definitions and adapter code. They receive the tasks from the **Workflow Worker**, execute them, and send the results back to the **Workflow Worker**. The Execution Workers are capable to load additional adapters as plugins, which allows them to work with different systems and technologies.
- **Adapters**: they interface and integrate with external systems, applications and technologies. Inside them, activities that can be consumed in a workflow are defined.
- **Adapter SDK**: a Software Development Kit that helps developers create new adapters to integrate with external systems.
- **Workflow Definitions**: workflow code written in the JSON format based on the Serverless Workflow specification.
- **K8s Infrastructure**: 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 Kubernetes cluster.
- **PostgreSQL**: it is the database used by the system to store and manage its data.

Check health and logs

CWM is a microservice-based application that leverages Kubernetes cluster architecture as its runtime environment. The health of the CWM application can thus be checked using Kubernetes commands.



Note To see all the supported `kubectl` commands, log in to the OS on your VM and use `kubectl --help`.

Check pod status

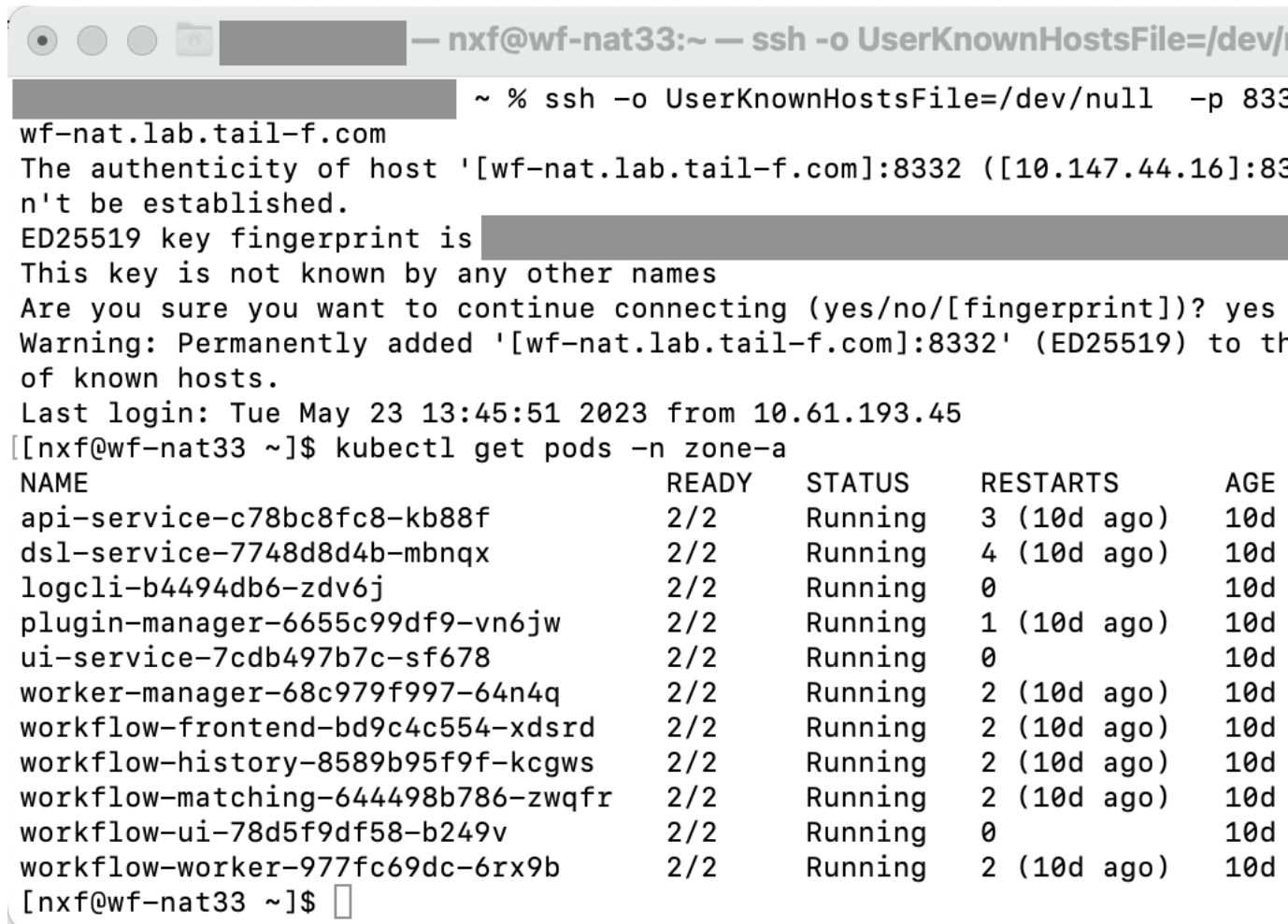
Step 1 Using a command-line terminal, log in to the OS on your virtual machine with SSH:

```
ssh -o UserKnownHostsFile=/dev/null -p 22 nxf@<your_resource_pool_address>
```

Step 2 To check status of pods for namespace `zone-a` (this is the default namespace for pods containing CWM microservices), run the following command:

```
kubectl get pods -n zone-a
```

Step 3 A list of pods will appear:



```

~ % ssh -o UserKnownHostsFile=/dev/null -p 8332 wf-nat.lab.tail-f.com
The authenticity of host '[wf-nat.lab.tail-f.com]:8332 ([10.147.44.16]:8332)' can't be established.
ED25519 key fingerprint is [redacted]
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '[wf-nat.lab.tail-f.com]:8332' (ED25519) to the list of known hosts.
Last login: Tue May 23 13:45:51 2023 from 10.61.193.45
[nxf@wf-nat33 ~]$ kubectl get pods -n zone-a
NAME                                READY   STATUS    RESTARTS   AGE
api-service-c78bc8fc8-kb88f         2/2     Running   3 (10d ago) 10d
dsl-service-7748d8d4b-mbnqx         2/2     Running   4 (10d ago) 10d
logcli-b4494db6-zdv6j               2/2     Running   0           10d
plugin-manager-6655c99df9-vn6jw     2/2     Running   1 (10d ago) 10d
ui-service-7cdb497b7c-sf678         2/2     Running   0           10d
worker-manager-68c979f997-64n4q     2/2     Running   2 (10d ago) 10d
workflow-frontend-bd9c4c554-xdsrd   2/2     Running   2 (10d ago) 10d
workflow-history-8589b95f9f-kcgws   2/2     Running   2 (10d ago) 10d
workflow-matching-644498b786-zwqfr  2/2     Running   2 (10d ago) 10d
workflow-ui-78d5f9df58-b249v        2/2     Running   0           10d
workflow-worker-977fc69dc-6rx9b     2/2     Running   2 (10d ago) 10d
[nxf@wf-nat33 ~]$

```

Step 4 If a pod has a status different from `Running`, you can 'restart' it using the following command:

```
kubectl delete pod <pod_name> -n zone-a
```

The pod will be deleted, but as Kubernetes configuration is declarative, it will effectively recreate the deleted pod and rerun it.

Check and collect logs

Application logs can be checked with **Loki logCLI** command-line interface. To gather logs from the CWM platform, follow these steps:

Step 1 Using a command-line terminal, connect to the system using SSH client:

```
ssh -pSSH_PORT nxf@ip_address_of_deployment
```

Note Adjust `SSH_PORT` and `ip_address_of_deployment` accordingly.

Step 2 After successful login, use the command below to list all running pods:

```
kubectl get pods -A
```

Example result:

```
[nxf@wf-nat-08 ~]$ kubectl get pods -A
```

NAMESPACE	NAME	READY	STATUS	RESTARTS
AGE				
kube-flannel	kube-flannel-ds-trr95	1/1	Running	0
103m				
kube-system	coredns-htg9j	1/1	Running	0
103m				
kube-system	etcd-wf-nat-08	1/1	Running	0
103m				
kube-system	kube-apiserver-wf-nat-08	1/1	Running	0
103m				
kube-system	kube-controller-manager-wf-nat-08	1/1	Running	0
103m				
kube-system	kube-proxy-c25f5	1/1	Running	0
103m				
kube-system	kube-scheduler-wf-nat-08	1/1	Running	0
103m				
local-path-storage	local-path-provisioner-6fb6f599c7-ckcjc	1/1	Running	0
103m				
nxf-system	authenticator-5db8885675-qlrmg	2/2	Running	0
102m				
nxf-system	controller-cbd87f8c5-6tg6f	2/2	Running	1 (102m ago)
102m				
nxf-system	ingress-proxy-56f7c9899d-6st6j	1/1	Running	0
102m				
nxf-system	kafka-0	1/1	Running	0
102m				
nxf-system	loki-7c994678f8-fnrs9	3/3	Running	0
102m				
nxf-system	minio-0	2/2	Running	0
103m				
nxf-system	postgres-0	2/2	Running	0
102m				
nxf-system	promtail-v6tb4	1/1	Running	0
102m				
nxf-system	registry-7dd84db44f-n5q7h	2/2	Running	0
102m				
nxf-system	vip-wf-nat-08-28131000-772k5	0/1	Completed	0
3m42s				
zone-a	api-service-745759bffc-v6r25	2/2	Running	2 (100m ago)
100m				
zone-a	dsl-service-77d5fc96cc-5nv42	2/2	Running	3 (100m ago)
100m				
zone-a	logcli-5c7ddbc95d-mkpcc	2/2	Running	0
100m				
zone-a	plugin-manager-665b7bbd4d-jvqdk	2/2	Running	1 (100m ago)
100m				
zone-a	ui-service-57cf6d6bcc-smmvt	2/2	Running	0
100m				
zone-a	worker-manager-6d6b445d46-r6nzk	2/2	Running	1 (99m ago)
100m				

zone-a 100m	workflow-frontend-77bc897549-kcz5k	2/2	Running	1 (99m ago)
zone-a 100m	workflow-history-58bdb85b8d-88t25	2/2	Running	1 (99m ago)
zone-a 100m	workflow-history-58bdb85b8d-h22bd	2/2	Running	1 (99m ago)
zone-a 100m	workflow-history-58bdb85b8d-ph5fh	2/2	Running	1 (99m ago)
zone-a 100m	workflow-matching-86cfc5577c-4mxhb	2/2	Running	1 (99m ago)
zone-a 100m	workflow-ui-68f857645-9mq9v	2/2	Running	0
zone-a 100m	workflow-worker-8496898f7b-wcrqs	2/2	Running	1 (99m ago)

Step 3 Identify the logcli tool available in the `zone-a` namespace. In this example, it is the pod named `logcli-5c7ddbc95d-mkpcc`.

Step 4 Connect to the correct pod and list the available log labels for filtering:

```
kubectl exec --namespace=zone-a -ti logcli-5c7ddbc95d-mkpcc -- logcli labels
app
container
filename
level
namespace
node_name
pod
stream
```

Step 5 Gather logs from all applications running in the "zone-a" namespace and save them to a single file. Make sure to adjust the `--since` option to collect logs from the relevant time period when the troubleshooting event occurred:

```
kubectl exec --namespace=zone-a -ti logcli-5c7ddbc95d-mkpcc -- logcli query '{namespace="zone-a"}'
--since 60m > zone-a.log
```

Step 6 Similarly, collect logs from other namespaces, using different files for convenience:

```
kubectl exec --namespace=zone-a -ti logcli-5c7ddbc95d-mkpcc -- logcli query '{namespace="nxf-system"}'
--since 60m > nxf-system.log

kubectl exec --namespace=zone-a -ti logcli-5c7ddbc95d-mkpcc -- logcli query '{namespace="kube-system"}'
--since 60m > kube-system.log
```

Step 7 Use the SCP tool to copy the log files from the system to your desktop:

```
scp -P SSH_PORT nxf@ip_address_of_deployment:"*.log".
```

Step 8 Finally, you can send the logs to support and provide a detailed description of the issue you are experiencing.

Note For more details on the logCLI commands and usage, refer to [logCLI Grafana documentation](#).



CHAPTER 3

API

This section contains the following topics:

- [CWM API Overview, on page 21](#)
- [Manage adapters, on page 22](#)
- [Manage workers, on page 24](#)
- [Manage workflows, on page 25](#)
- [Manage resources and secrets, on page 25](#)
- [Manage Schedules, on page 27](#)
- [Manage Jobs, on page 30](#)
- [Manage Event types, on page 33](#)

CWM API Overview

The CWM API was developed according to the Representational State Transfer (REST) design principles. The API is accessed using HTTP with JSON data format. The success or failure of the request is indicated by the relevant HTTP response code. Data retrieval methods require a GET request, while methods for adding, changing, or deleting data require POST, PUT, PATCH, or DELETE methods. Errors will be returned if the request is sent with the wrong request type.

How to use CWM API?

You can consume CWM API in two ways:

- via Swagger interface, or
- via [Postman collection](#).

Built directly into the product is a Swagger interface accessed from the CWM UI, but for ease of use, a Postman collection with example requests is also provided.

All tutorials under *Manage via API* section assume the use of Swagger.

Use Swagger

To access CWM Swagger API, from the navigation menu on the left, click the **swagger** icon.

Use CWM Postman collection

Prerequisites

- Postman Web app account or Postman Desktop installed.

Download JSON collection file

Download the [Postman collection in JSON format by clicking this link](#). Unpack the zip archive.

Import collection and set environment

-
- Step 1** Open 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 have unpacked from the zip archive.
 - Step 3** Go to **Environments** and select the newly imported **test** environment.
 - Step 4** Provide current values for the **baseUrl** and **port** variables to fit your CWM IP address and port and save the changes.
- Now you're set up and ready to use the collection.
-

Manage adapters

To interact with external target systems, CWM requires adapters. You can manage them in the CWM UI as described in the **Operator** guide, or using the CWM API. The following API endpoints are available for handling adapters:

- `GET/adapters`: gets a list of adapters existing in the CWM application.
- `POST/adapters`: uploads an adapter **.tar** file to CWM storage.
- `GET/adapters/{adapterId}`: gets the details of a specific adapter existing in the CWM application. Among others, it lists all the activities available in the adapter.
- `PUT/adapters/{adapterId}`: updates an existing adapter file with a new adapter version.
- `DELETE/adapters/{adapterId}`: deletes an adapter from the CWM application.
- `POST/adapters/{adapterId}/deploy`: deploys an adapter in the system based on the uploaded adapter file.
- `PATCH/adapters/{adapterId}`: updates the default status of the adapter.

Install adapter

CWM adapters come in **.tar** installation files. Before they can be used in a workflow, they need to be uploaded to storage and deployed in the system. Here's how to do it.

Upload adapter file

Before you deploy an adapter, you need to upload the adapter **.tar** file to CWM storage:

-
- Step 1** Get a latest adapter installation file or create your own adapter.
 - Step 2** Log in to CWM and from the navigation menu on the left, click the `:simple-swagger:` icon.
 - Step 3** In the **adapters** section, click the `POST/adapter` endpoint to expand it. Inside the endpoint, click **Try it out**.
 - Step 4** In the subsection that appears, click **Choose File**, select the adapter **.tar** installation file and click **Upload**, then click **Execute**.

If the server response code is 201, the adapter file is successfully uploaded into the CWM database.

Deploy adapter

-
- Step 1** In the CWM API **adapters** section, click the `GET/adapter` endpoint to expand it. Inside the endpoint, click **Try it out** and **Execute**.
 - Step 2** From the server response body, copy the value of the `id` field for your uploaded adapter.
 - Step 3** In the CWM API **adapters** section, click the `POST/adapter/{adapterId}/deploy` endpoint to expand it.
 - Step 4** Inside the endpoint, click **Try it out**. Paste the adapter id into the **Adapter ID** field.
 - Step 5** In the **createWorker** field, you may set the `createWorker` parameter to `true`. This will [create a worker](#) with the same name as the adapter id.
 - Step 6** Click **Execute**.

If the server response code is 201, the adapter plugin is successfully installed and you're good to proceed.

Delete adapter

To delete an adapter permanently from storage and uninstall it:

-
- Step 1** In the CWM API **adapters** section, click the `GET/adapter` endpoint to expand it. Inside the endpoint, click **Try it out** and **Execute**.
 - Step 2** From the server response body, copy the value of the `id` field for your uploaded adapter.
 - Step 3** In the CWM API **adapters** section, click the `DELETE/adapter/{adapterId}` endpoint to expand it.
 - Step 4** Inside the endpoint, click **Try it out**. Paste the adapter id into the **Adapter ID** field.
 - Step 5** Click **Execute**.
-

Manage workers

Workers are processes that execute actions defined in workflow definitions and adapter code. You can manage them using the CWM UI as described in the **Operator** guide, or with the CWM API, as described below.

The following actions for managing workers are available:

- `GET/worker`: gets a list of workers existing in the CWM application.
- `POST/worker`: creates a new worker in the CWM application.
- `GET/worker/{workerName}`: gets the details of a specific worker existing in the CWM application.
- `PUT/worker/{workerName}`: updates an existing worker with new parameter values.
- `DELETE/worker/{workerName}`: deletes a worker from the CWM application.
- `POST/worker/{workerName}/start`: activates a worker created in the application.
- `POST/worker/{workerName}/stop`: deactivates a worker created in the application.

Create worker

-
- Step 1** Log in to CWM and from the navigation menu on the left, click the **swagger** icon.
- Step 2** In the CWM API **workers** section, click the `POST/worker` endpoint to expand it. Inside the endpoint, click **Try it out**.
- Step 3** In the **Worker data** field, provide the required values:
- "activities": paste the ID of your deployed adapter or specific adapter activity.
 - "startWorker": set to `true`.
 - "workerName": provide a name for your worker.
- Step 4** Click **Execute**.
-

Start a worker

-
- Step 1** In the CWM API **workers** section, click the `POST/{workerName}/start` endpoint to expand it. Inside the endpoint, click **Try it out**.
- Step 2** In the **parameters** fields, provide the required values:
- "Name of a worker to start": paste the name the worker to be started.
 - "forceReload": set to `true` if you want to force the worker to start.
- Step 3** Click **Execute**.
-

Stop a worker

-
- Step 1** In the CWM API **workers** section, click the `POST/{workerName}/stop` endpoint to expand it. Inside the endpoint, click **Try it out**.
- Step 2** In the **parameters** fields, provide the required values:
- a) "Name of a worker to stop": paste the name the worker to be stopped.
 - b) "forceStop": set to `true` if you want to force the worker to stop.
- Step 3** Click **Execute**.
-

Manage workflows

Workflow definitions can be managed both in the CWM UI as described in the **Operator** guide, or using the CWM API:

- `GET/workflow`: gets a list of workflow definitions existing in the CWM application.
- `POST/workflow`: creates a new workflow definition in the CWM application.
- `GET/workflow/{workflowId}`: gets the details of a specific workflow existing in the CWM application.
- `PUT/workflow/{workflowId}`: updates an existing workflow definition in the CWM application.
- `DELETE/workflow/{workflowId}`: deletes a selected workflow definition from the CWM application.
- `GET/workflowExport`: exports workflow definitions based on a given array of workflow definition IDs.
- `POST/workflowImport`: imports in bulk workflow definitions to the CWM application.



Note The recommended method for managing workflows is through CWM UI. For details, refer to the **Operator** guide.

Manage resources and secrets

Overview

For CWM, adapters define activities that enable to execute actions in external entities, such as other systems or applications. These entities are, in most cases, integrated via APIs which usually require connection and authentication data. CWM provides a framework where when an activity is consumed in a workflow, the details of a connection endpoint and authentication data can be passed at runtime. Thus, the operator who runs a workflow may not know any details of these systems (resources) such as IP addresses, ports or usernames and passwords.

CWM provides a framework for secure handling of resources and secrets in the database and identifying them by their respective IDs. When running a workflow, just the resource ID needs to be passed, with the rest of the data sent to the adapter by the Resource Manager without any intervention from the Operator or additional development from Adapter developer. You can manage secrets and resources in the CWM UI as described in the **Operator** guide, or using the CWM API.

Resource and secret types

You can think of resource and secret types as buckets used to organize resources and secrets created by users by their type. Types are defined inside a given adapter and are added to the system automatically upon installing the adapter. You can list secrets belonging to a specific type using the `GET/secretType/{secretTypeId}` API endpoint.

Secrets API endpoints

The following actions for managing secrets are available:

- `GET/secret`: gets a list of secrets existing in the CWM application.
- `POST/secret`: creates a new secret in the CWM application.
- `GET/secretType/{secretTypeId}`: lists secrets existing in the CWM application that belong to a specific type.
- `GET/secretType`: gets a list of secret types existing in the CWM application.
- `GET/secret/{secretId}`: gets details of an existing secret.
- `DELETE/secret/{secretId}`: deletes a secret from the CWM application.
- `PATCH/secret/{secretId}`: updates a secret existing in the CWM application with new parameter values.

Resources API endpoints

The following actions for managing resources are available:

- `GET/resource`: gets a list of resources existing in the CWM application.
- `POST/resource`: creates a new resource in the CWM application.
- `GET/resource/{resourceId}`: gets the details of a specific resource existing in the CWM application.
- `PATCH/resource/{resourceId}`: updates an existing resource with new parameter values.
- `DELETE/resource/{resourceId}`: deletes a resource from the CWM application.
- `GET/resourceType`: gets a list of resource types existing in the CWM application.
- `GET/resourceType/{resourceTypeId}`: gets the details of an existing resource type.

Create secret

-
- Step 1** Log in to CWM and from the navigation menu on the left, click the `:simple-swagger:` icon.
- Step 2** In the CWM API **secrets** section, click the `POST /secret` endpoint to expand it.

Step 3 Inside the endpoint, click **Try it out**, and provide your data into the **Secret input** field. Example input can look like this:

```
{
  "secret": {
    "username": "admin",
    "password": "admin"
  },
  "secretId": "NSOSecret",
  "secretType": "basicAuth"
}
```

Step 4 Click **Execute**.

If the server response code is 201, the secret is successfully created and you can start creating a resource to associate the secret with.

Create resource

Step 1 In the CWM API **resources** section, click the `POST /resource` endpoint to expand it.

Step 2 Inside the endpoint, click **Try it out**, and provide your data into the **Resource input** field. Example input can look like this:

```
{
  "resource": {
    "scheme": "http",
    "host": "127.0.0.1",
    "port": 8080
  },
  "resourceId": "NSOLocal",
  "resourceType": "cisco.nso.resource.v1.0.0",
  "secretId": "NSOSecret"
}
```

Step 3 Click **Execute**.

If the server response code is 201, the resource is successfully created.

Manage Schedules

To automate recurring operations or settle them on a specific date and time in the future, you can create a schedule. In CWM, there are two types of schedules:

- One-time: define at what time and date a single job will be executed
- Recurring: define rules (based on the interval, calendar or cron expression) when the job run repeats

CWM scheduler API allows you to create, update and pause/unpause schedules. While creating a basic schedule (along with other operations like deleting a schedule) in `{{ version.CWM }}` is possible via UI, defining advanced schedules using more functionalities of the scheduler is available only via API.

The following API endpoints are available for handling schedules:

- `GET/schedule`: gets a list of schedules existing in the CWM application.
- `POST/schedule`: creates a new schedule in the CWM application.
- `GET/schedule/{scheduleId}`: gets the details of a specific schedule existing in the CWM application.
- `PATCH/schedule/{scheduleId}`: updates an existing schedule with new detail(s).
- `DELETE/schedule/{scheduleId}`: deletes a schedule from the CWM application.

Create a schedule

To create a schedule, follow the steps below:

-
- Step 1** Log in to CWM and from the navigation menu on the left, click the `:simple-swagger:` icon.
 - Step 2** In the CWM API **scheduler** section, click the `POST/schedule` endpoint to expand it.
 - Step 3** Inside the endpoint, click **Try it out** and in the **Schedule request** provide the chosen values.
 - Step 4** Click **Execute**. If the server response code is 201, the schedule has been successfully created.
-

Update a schedule

To edit a schedule, follow the steps below:

-
- Step 1** Log in to CWM and from the navigation menu on the left, click the `:simple-swagger:` icon.
 - Step 2** In the CWM API **scheduler** section, click the `PATCH/schedule` endpoint to expand it.
 - Step 3** Inside the endpoint, click **Try it out**, provide the *Schedule ID* and in the **Schedule update request** edit the chosen values.
 - Step 4** Click **Execute**. If the server response code is 200, the schedule has been successfully updated.
- Updating a schedule replaces the entire configuration. It means that if you want to change only one existing value, you still need to pass all of the details again, even if they will be the same.
-

Pause a schedule

You can pause a schedule for a chosen time, for example, for a maintenance window. When the schedule is paused, the runs that are supposed to be executed are skipped. To pause a schedule, follow the steps below:

-
- Step 1** Log in to CWM and from the navigation menu on the left, click the `:simple-swagger:` icon.
 - Step 2** In the CWM API **scheduler** section, click the `PATCH/schedule` endpoint to expand it.
 - Step 3** Inside the endpoint, click **Try it out**, provide the *Schedule ID* and in the **Schedule update request** set the value of the field "paused" to `true`.
 - Step 4** Click **Execute**. If the server response code is 200, the schedule has been successfully paused.

The schedule remains paused as long as you resume it with the next request.

Unpause a schedule

To resume a schedule, follow the steps below:

- Step 1** Log in to CWM and from the navigation menu on the left, click the swagger icon.
- Step 2** In the CWM API **scheduler** section, click the `PATCH/schedule` endpoint to expand it.
- Step 3** Inside the endpoint, click **Try it out**, provide the *Schedule ID* and in the **Schedule update request** set the value of the field "paused" to `false`.
- Step 4** Click **Execute**. If the server response code is 200, the schedule has been successfully resumed.

Pause on failure

If you set `pauseOnFailure` field to `true`, the schedule will be automatically paused after any of its job fails. It gives a chance to address the issue, for example, when the workflow definition associated with the schedule will be deleted. To change the pause on failure value, follow the general steps in **Update a schedule**.

Change the overlap policy

Overlap policy controls what happens when the next job should be started by a schedule at the same time that a previous run is still being executed. The default policy is **Skip** (with the "overlap" field value set to 1), which means that when the previous job is still running, the next scheduled run won't start and it will be skipped. When the existing run completes, only the next scheduled run after that time will be considered. To change the overlap policy, follow the general steps in **Update a schedule**.

Possible policies:

Policy type	overlap field value	Description
Skip	1	This is the default policy that prevents overlapping runs. While the previous run from a schedule is still running when the next one should be executed, the next run will be skipped.
Buffer One	2	Starts the next run as soon as the current one completes. Only one run will be buffered. If there are more runs that are supposed to happen when the current job is running, they are skipped, and only the first one in the queue will be executed after the running job finishes.
Buffer All	3	Buffers all runs that are supposed to happen when the current job is running. All buffered runs will be executed sequentially, directly after the running job completes.
Cancel Other	4	Cancels the running job and starts the new one after the cancellation of the old one is completed.

Policy type	overlap field value	Description
Terminate Other	5	Terminates running job and starts the new one immediately.
Allow All	6	Starts any number of concurrent runs.

Manage Jobs

Jobs can be managed both in the CWM UI as described in the Operator guide or using the CWM API.

!!! note "Important" Some functionalities, for example, querying the jobs based on multiple tags, for {{ version.CWM }} are available only via API.

The following actions for managing jobs are available:

- `GET/job`: gets a list of jobs existing in the CWM application.
- `POST/job`: creates a new job run in the CWM application based on given parameters.
- `GET/job/{jobId}/runs/{runId}`: returns the details of a specific job existing in the CWM application.
- `POST/job/{jobId}/runs/{runId}/cancel`: cancels the execution of a running job, after the workflow worker completes the ongoing task execution from the workflow definition.
- `GET/job/{jobId}/runs/{runId}/events`: returns the event history for a specific job.
- `POST/job/{jobId}/runs/{runId}/terminate`: immediately terminates the execution of a running job.

Execute a job

To run a job, follow the steps below:

-
- Step 1** Log in to CWM and from the navigation menu on the left, click the :simple-swagger: icon.
- Step 2** In the CWM API **jobs** section, click the `POST/job` endpoint to expand it.
- Step 3** Inside the endpoint, click **Try it out** and in the **Job Execution Request** provide the chosen values, for example:

```
{
  "data": {},
  "jobName": "test API job",
  "tags": [
    "test", "API"
  ],
  "workflowName": "Test cisco workflow",
  "workflowVersion": "1.1"
}
```

Note Job tags are optional but may ease filtering specific jobs in the future.

- Step 4** Click **Execute**.

If the job run has been successfully created, you should get a server response with a code 200 and the job ID and run ID returned.

Filter jobs by multiple tags

You can get a list of jobs existing in CWM filtered by specific queries, for example, by associated tags. To get a list, follow the steps below:

- Step 1** Log in to CWM and from the navigation menu on the left, click the :simple-swagger: icon.
- Step 2** In the CWM API **jobs** section, click the `GET/job/{jobId}/runs/{runId}` endpoint to expand it.
- Step 3** Inside the endpoint, click **Try it out** and in the `query` parameter, specify the tags by which you want to filter the jobs, following the schema: `JobTags = "tag_name1"` and `JobTags = "tag_name2"`, as in the example below:

Figure 6: Job Event Log

Parameters

Name	Description
pageSize integer (<i>query</i>)	Number of jobs to return in each page <input type="text" value="pageSize"/>
nextPageToken string (<i>query</i>)	Page token to fetch the next set of results <input type="text" value="nextPageToken"/>
query string (<i>query</i>)	The query to filter jobs by <input type="text" value='JobTags = "cisco" and JobTags = "NSO"'/>

Execute

Step 4 Click **Execute**.

You should receive a server response with a 200 status code, along with the filtered jobs details.

Known issue: In the response body with the job details, the field named `workflowId` is actually a Job ID, not a Workflow definition ID.

Get event history

To get a list of event history (all or filtered) for a given job, follow the steps below:

Step 1 Log in to CWM and from the navigation menu on the left, click the **swagger** icon.

Step 2 In the CWM API **jobs** section, click the `GET/job/{jobId}/runs/{runId}/events` endpoint to expand it.

Step 3 Inside the endpoint, click **Try it out** and provide the Run ID and Job ID of the job for which you want to get the event history.

Step 4 Optionally, you can set `isLongPoll` parameter to `true` if you are querying the currently running job. Then, the connection will be open until the execution of a given job finishes, and you will get the response after the job execution is completed. If you set it to `false`, you will receive a history of events immediately after sending the request, consisting of events completed up to the moment of request.

Step 5 Optionally, you can set `filterType` parameter to the chosen value (`all` or `close_event`).

Note The `close_event` value filters only for the closing event of an already finished job, for example `WorkflowExecutionFailed` or `WorkflowExecutionCompleted`. If you picked the `close_event` as a filter, and your job is currently running, you will receive a 400 error.

Step 6 Click **Execute**.

You should receive a server response with a 200 status code, along with the filtered events.

Manage Event types

Event types are categories of signals either received or produced by CWM and referred to in workflow definitions. You can manage them using the CWM UI as described in the **Operator** guide, or with the CWM API, as described below.

The following actions for managing Event types are available:

- `GET/eventType`: gets a list of Event types existing in the CWM application.
- `POST/eventType`: creates a new Event type in the CWM application.
- `GET/eventType/{name}`: gets the details of a specific Event type existing in the CWM application.
- `PUT/eventType/{name}`: updates an existing Event type with new parameter values.
- `DELETE/eventType/{name}`: deletes a Event type from the CWM application.
- `POST/eventType/{name}/start`: starts or stops an event listener.

Create Event type

-
- Step 1** Log in to CWM and from the navigation menu on the left, click the `:simple-swagger:` icon.
- Step 2** In the CWM API **eventType** section, click the `POST/eventType` endpoint to expand it. Inside the endpoint, click **Try it out**.
- Step 3** In the **eventType data** field, modify the required values:
- ```
{
 "correlation": [
 {
 "contextAttributeName": "string",
 "contextAttributeValue": "string"
 }
],
 "createWorkflow": false,
 "dataOnly": true,
 "endpoint": "string",
 "kind": "string",
 "name": "string",
 "resourceId": "string",
 "source": "string",
 "type": "string",
 "workflowName": "string",
 "workflowVersion": "string"
}
```
- Step 4** Click **Execute**.
- 

## Start/stop an Event type listener

- 
- Step 1** In the CWM API **eventType** section, click the `POST/{name}/{action}` endpoint to expand it. Inside the endpoint, click **Try it out**.
- Step 2** In the **parameters** fields, provide the required values:
- "Name": paste the name the Event type for which a listener needs to be started/stopped.
  - "action": set to `start` if you want to start a listener, or `stop` if you want to stop a running listener.
- Step 3** Click **Execute**.
-



## CHAPTER 4

# Users

---

This section contains the following topics:

- [Manage user access, on page 35](#)

## Manage user access

In CWM, you can manage user access via NxF which adds a layer of security and works as a Single Authentication Agent, thus sharing local, LDAP, and SAML users.

## NxF functionality in CWM

NxF functionality is available for admin users from the **Settings** tab in the CWM UI. To access NxF functionality in CWM:

- 
- Step 1** In CWM, go to the outermost navigation menu on the left.
- Step 2** Click the **Settings** icon.

*Figure 7: NxF Settings*

**Step 3** In the expanded drawer, you can find the following:

*Figure 8: NxF Drawer Settings*



- a) A) **System Info** section with information about the latest versions of NxF and CWM microservices.
  - b) B) **Security** section for access management:
    - **Local Users**: where you can display, create and edit local users via UI.
    - **LDAP**: where you can set LDAP settings for user authentication.
    - **SAML SSO**: where you can set SAML Single-Sign-On settings for user authentication.
    - **Permission Mapping**: where you can handle permission management via Cisco Policy Management Tool.
- 

## Add local user

---

- Step 1** In CWM, go to the outermost navigation menu on the left.
- Step 2** Navigate to **CWM** (Cisco icon) -> **Local Users** tab.
- Step 3** Click **Add...**
- Step 4** In the Add User panel, fill in the mandatory fields (marked with an asterisk): Username (used to log in to the CWM), Password, Confirm Password and Access Permissions (enter `permission/user`). The Description and Display Name (visible next to the username in CWM) are optional fields.

*Figure 9: NxF Add User*

**Step 5** Use radio buttons to set the user status. You can make both radio buttons disabled or enabled at the same time.

- a) **Active enabled**: allows the user to log in to the CWM.
- b) **Active disabled**: forbids the user to log in to the CWM.
- c) **Locked enabled**: prevents deleting the user.
- d) **Locked disabled**: allows removal of the user.

**Step 6** Click **Save**.

---

## Set up authentication via LDAP

Besides supporting local users, CWM allows adding LDAP users through integration with LDAP (Lightweight Directory Access Protocol) servers.

---

**Step 1** In CWM, go to the outermost navigation menu on the left.

**Step 2** Navigate to **CWM** (Cisco icon) -> **LDAP** tab.

**Step 3** Click the **Enabled** radio button.

**Step 4** Fill in the mandatory fields (marked with an asterisk): LDAP Server Address, Bind DN, Bind Credentials and Search Filter. Search Base and Root CAs are optional.

*Figure 10: NxFLDAP*

**Step 5** Click **Save**.

---

## Set up authentication via SAML SSO

CWM offers SAML SSO feature that supports both LDAP and non-LDAP users to gain single sign-on access based on the protocol SAML (Security Assertion Markup Language). You can enable SAML SSO for CWM along with LDAP or without it.

---

**Step 1** In CWM, go to the outermost navigation menu on the left.

**Step 2** Navigate to **CWM** (Cisco icon) -> **SAML SSO** tab.

**Step 3** Click the **Enabled** radio button.

**Step 4** Fill in the mandatory fields: Login URL, Entity ID, Base URL, Signing Certificate and Groups Attribute Name.

*Figure 11: NxF SAMLSSO*

**Step 5** Click **Save**.

---

## Set up permission mapping

You can give specific permissions to a group of users via Cisco Policy Management Tool (PMT).

---

**Step 1** In CWM, go to the outermost navigation menu on the left.

**Step 2** Navigate to **CWM** (Cisco icon) -> **Permission Mapping** tab.

**Step 3** Click **Add...**

**Step 4** In the Add Permission Mapping panel, choose one **Mapping Type** from the dropdown menu: SAML User, SAML Group, LDAP User, or LDAP Group.

Figure 12: NxF Permission Mapping

SYSTEM INFO

Versions

SECURITY

Local Users

LDAP

SAML SSO

Permission Mapping

## Add Permissions

Mapping Type\*

SAML Group

Match\*

crosswork-workflow

Access Permission\*

permission/admin

- Step 5** Fill in the Match field with the entry from the Cisco Policy Management Tool. You can find the match in PMT UI -> **OAuth Clients** tab -> Client ID Column.
- Step 6** Enter appropriate permission (for example `permission/admin`) in the Access Permission field.
- Step 7** Click **Save**.





## CHAPTER 5

# Adapters

---

This section contains the following topics:

- [Use generic-email adapter, on page 47](#)

## Use generic-email adapter

The **Email Adapter** (generic-email) adds an element of reporting into your workflows by providing basic functionality to send emails using an SMTP server. For the 1.0.0 adapter version, you can use the `Send` activity to send an email with a message defined inside the workflow definition.

## Get generic-email adapter

Download the CWM 1.1 Software package. The `cwm.v1.1.generic.email.v1.0.0.tar.gz` file is included inside the package.

## Install adapter

To install the adapter, follow the instructions on how to install an adapter in the Operator guide.

## Create SMTP resource and secret

Before going into the details of defining your email message inside a workflow, you need to add a resource and a secret to CWM. You will later need to reference them inside your workflow.

## Add secret

- 
- |               |                                                                                                                                                           |
|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Step 1</b> | In CWM, navigate to the <b>Admin</b> -> <b>Secrets</b> tab.                                                                                               |
| <b>Step 2</b> | Click <b>Add Secret</b> .                                                                                                                                 |
| <b>Step 3</b> | In the <b>New secret</b> view, specify the following:                                                                                                     |
|               | a) Secret ID: name your secret. You'll need to reference this secret ID later in the resource and inside the workflow.<br>E.g. <code>emailSecret</code> . |
|               | b) Secret type: select <code>basicAuth</code> .                                                                                                           |

- Step 4** After selecting the secret type, a set of additional fields is displayed under the Secret type details section. Fill in the fields with the following:
- a) password: provide password to your sender email address.
  - b) username: provide the address you will send the email from in the format `sender@address.com`.
- Step 5** Click **Create Secret**.

## Add resource

- Step 1** In CWM, navigate to the **Admin -> Resources** tab.
- Step 2** Click **Add Resource**.
- Step 3** In the **New resource** window, specify the following:
- a) Resource name: name your resource. You'll need to provide the resource ID later inside the workflow as a reference.  
E.g. `emailResource`.
  - b) Resource type: select `generic.email.resource.v1.0.0`.
  - c) Secret ID: provide the ID of the secret you've just added.
  - d) Connection:
    - Host: provide the address of the SMTP server to be used.
    - Port: provide the SMTP port. The standard SMTP ports for encrypted email transmissions are either 587 or 25.
    - Scheme: this field is not required.
    - Timeout: this field is not required.
    - Allow Insecure: select `true`.
- Step 4** Click **Create resource**.

## Define the Send activity in workflow

Learn how to use the adapter Send activity in a workflow.

### Set activity reference

In CWM, the adapter Send activity is referred to as `generic.email.smtp.Send`. When defining a workflow, you need to specify it as the value of the `operations` parameter under `functions`:

```
"functions": [
 {
 "name": "smtp.send",
 "operation": "generic.email.v1.0.0.smtp.Send"
 }
]
```



**Note** Inside the `name` parameter, provide an activity name that you will later refer to in the `refName` parameter while defining the action.

## Define email message in actions

Now you can define an action in which an email will be sent as part of a workflow state.

The available input parameters for the action are:

| Field   | Type   | Label    | Description                           |
|---------|--------|----------|---------------------------------------|
| from    | string |          | Sender email address                  |
| to      | string | repeated | List of recipient email addresses     |
| cc      | string | repeated | List of recipient cc email addresses  |
| bcc     | string | repeated | List of recipient bcc email addresses |
| subject | string |          | Email title                           |
| text    | string |          | Email body as text                    |
| html    | string |          | Email body as html                    |

Use the available fields as the `input` key/value pairs within the `arguments` that define the `SendEmail` example action as shown below:

```
"states": [
 {
 "name": "EmailState",
 "type": "operation",
 "end": true,
 "actions": [
 {
 "name": "SendEmail",
 "functionRef": {
 "refName": "smtp.send",
 "arguments": {
 "input": {
 "to": ["recipient1@address.com", "recipient2@address.com"],
 "from": "sender@address.com",
 "text": "Hello, this is some placeholder email text.",
 "subject": "A test email from CWM"
 },
 "config": {
 "resourceId": "emailResource"
 }
 }
 }
 }
]
 }
]
```

If you want to trigger the email action based on a condition, you can use the `Switch` state and define the `dataConditions` parameter inside it. For details, check out the [Serverless Workflow Specification documentation](#) for the `Switch` state.





## CHAPTER 6

# Events

---

This section contains the following topics:

- [Event handling overview, on page 51](#)
- [Define a Kafka event, on page 58](#)

## 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. On top of that, for each event type that you define in CWM, 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 the event format section for more details.

## Brokers and protocols

Crosswork Workflow Manager 1.1 supports the Kafka broker and the AMQP and HTTP protocols for handling events. Events can be either **consume** by a workflow running inside CWM (incoming events forwarded by a broker) or **produce** by a running workflow and forwarded to an external system (outgoing events received by a broker).



---

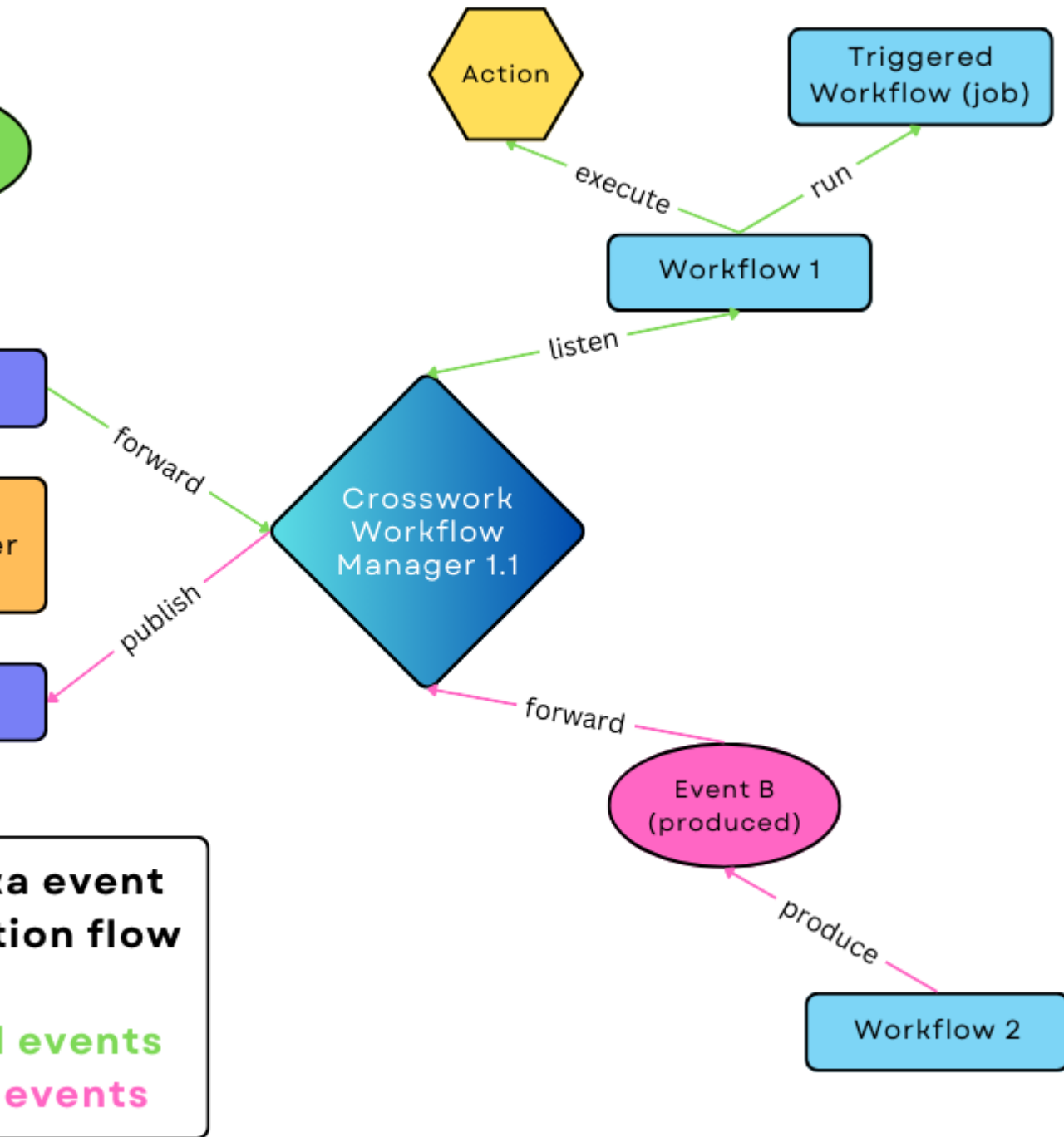
**Note** It is important to remember that CWM doesn't act as an event broker itself. It provides means to connect to external brokers to forward messages/events.

---

## Kafka broker

For **consume** event kind, 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. Then, the workflow executes actions defined inside the Event State and/or runs another workflow execution (if selected).

For **produce** event kind, 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



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

## AMQP protocol (e.g. RabbitMQ broker)

For **consume** event kind, 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. Then, the workflow executes actions defined inside the Event State and/or runs another workflow execution (if selected).

For **produce** event kind, 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 list of supported formats is here:  
<https://github.com/cloudevents/spec?tab=readme-ov-file>.

## HTTP protocol

For **consume** event kind, CWM exposes an HTTP endpoint that listens for any incoming events. If an event of 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 message body needs to be in JSON format representing a Cloud Event:




---

**Note** 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=\"xml\"/>", "contextAttrName": "contextAttrValue" }

---

For **produce** kind 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

resource

⌚ Last Refresh: 10-Apr-2024 12:55:34 PM CEST | ↻

Cancel

Create resource

## Connection

Url\*

http://example.com

Method

PUT

Headers

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

## Event system configuration

### Secret

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. To learn how to do it, see the section on adding secrets.

## Resource

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**: provide your Kafka version. The standard way to check Kafka version is to run `bin/kafka-topics.sh --version` in a terminal.
  - **Brokers**: provide 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 if needed.
- For HTTP:
  - **Produce** event kind: fill in the **URL** field and optionally, **Method** and **Headers** (for example, Client-ID header name and value as a JSON object).



### Note

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

- **Consume** event kind: 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 later use the URL path as the **End point** when configuring the event type.

## Event type



**Note** 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 HTTP **consume** event kind, 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 `{{ version.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**, provide attribute name and value, and click

Selected 0 / Total 1 



## Actions

 Edit

Delete



### 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 need to follow the [Cloud Events specification](#) format. Minimum viable event message contains 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 like "datacontenttype", "data", and 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 that the workflow will be waiting on. These are:

- **Event definition:** used to define the event type and its properties:

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

- **Event state:** used to define actions to be taken when the event occurs:

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

## Define a Kafka event

### Prerequisites

- A set-up Kafka service (or RabbitMQ with AMQP 1.0 plugin in case of AMQP, or any HTTP Client).
- CWM 1.1 installed using OVA.

### Step 1: Create Kafka secret and resource

To enable a secure connection to the Kafka service, you need to create a secret with Kafka credentials and a resource with connection details. Here's how to do it:

## Create secret

- 
- Step 1** In CWM, navigate to the **Admin** -> **Secrets** tab.
- Step 2** Click **Add Secret**.
- Step 3** In the **New secret** view, specify the following:
- a) Secret ID: `KafkaSecret`
  - b) Secret type: `basicAuth`
- Step 4** After selecting the secret type, a set of additional fields is displayed under the Secret type details section. Fill in the fields:
- a) password: password used for logging in to Kafka.
  - b) username: username used for logging in to Kafka.
- Step 5** Click **Create Secret**.
- 

## Create resource

- 
- Step 1** In CWM, navigate to the **Admin** -> **Resources** tab.
- Step 2** Click **Add Resource**.
- Step 3** In the **New resource** window, specify the following:
- a) Resource name: `KafkaResource`
  - b) Resource type: `cisco.cwm.kafka.v1.0.0` (or `cisco.cwm.amqp.v1.0.0` or `cisco.cwm.http.v1.0.0` if you use these protocols instead)
  - c) Secret ID: `KafkaSecret`
  - d) Connection:
    - **KafkaVersion**: provide your Kafka version. The standard way to check this is to run `bin/kafka-topics.sh --version` in a terminal.
    - **Brokers**: provide your Kafka broker address in the following format: `["localhost:9092"]`.
    - **OtherSettings**: an editable list with default Kafka setting values. You can modify the values if needed.

**Note** Connection settings differ in case of **AMQP** and **HTTP** resource types:

- For AMQP, provide the **ServerDNS** in the following format: ``amqp://localhost:5723``.
- For HTTP, provide the **URL** and additional **headers** (for example, Client-ID header name and value). Note that URL needs to be your host address but without the URL path. This you will specify as **End point** when configuring the resource type.

**Step 4** Click **Create resource**.

## Step 2: Add event type to CWM

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

**Step 1** In the CWM UI, select the **Admin** tile from the navigation menu on the left.

**Step 2** In the **Event system** panel, click **Add event type**.

**Step 3** In the **New event type** modal, provide the required input:

- a) **Event type name:** provide name for your event type. You will later refer to it inside the workflow definition.
- b) **Resource:** from the list, select `KafkaResource`.
- c) **Event source:** define your event source. It's fully user-defined and will be referenced in the workflow definition. Required for `produce` event kind.
- d) **End point:** for Kafka, provide your Kafka topic (event stream). For AMQP, provide endpoint (terminus). For HTTP, provide URL (Host) path.
- e) **Select kind:** from the list, select `consume`.

**Note** Use `Produce` to define an event to be produced by a workflow and consumed by another system. In this case, the remaining **Step 2** settings presented below this point won't apply. The `both` option is not yet supported for CWM 1.1.

- f) **Start listener:** click it to start listening for the defined event type.
- g) **Run job:** tick this checkbox if you want to trigger a workflow upon receiving the event. Then select the desired workflow from the list.
- h) **Correlation context 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 filter that refines the inbound event data and triggers actions defined inside a "listening" workflow on the basis of your context attributes.
- i) Click **Add attribute** and provide attribute name and value (fully user-defined).

**Step 4** Click **Create Event type**.

## Step 3: Define 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. For this purpose, we'll need to define the event using an [Event definition](#) and specify the [Event state](#) and define actions to be taken when the event occurs. For example purposes, let's take a scenario where a router overheating alarm (inbound event) triggers the workflow event state and two remediation actions are executed:

```
{
 "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": "${ .RouterIP }"
 },
 "resultEventTimeout": "PT30M"
 }
 },
 {
 "functionRef": {
 "refName": "MoveTraffic",
 "contextAttributes": {
 "RouterIP": "${ .RouterIP }"
 },
 "resultEventTimeout": "PT30M"
 }
 }
],
 "eventRefs": ["HighRouterTemp"],
 "timeouts": {
 "actionExecTimeout": "PT60M"
 }
 }
]
 },
 {
 "version": "1.0.0",
 "description": "Remediate router overheating",
 "specVersion": "0.8"
 }
]

```



**Note** Note that the example is not a complete workflow. It presents a sample of how you can define an event inside a workflow and act on it.