

**Intelligent EdgeFabric**

# **User Guide**

**Issue**            01  
**Date**             2023-11-23



**Copyright © Huawei Technologies Co., Ltd. 2024. All rights reserved.**

No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Huawei Technologies Co., Ltd.

## **Trademarks and Permissions**



HUAWEI and other Huawei trademarks are trademarks of Huawei Technologies Co., Ltd.

All other trademarks and trade names mentioned in this document are the property of their respective holders.

## **Notice**

The purchased products, services and features are stipulated by the contract made between Huawei and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

# Security Declaration

## Vulnerability

Huawei's regulations on product vulnerability management are subject to the *Vul. Response Process*. For details about this process, visit the following web page:

<https://www.huawei.com/en/psirt/vul-response-process>

For vulnerability information, enterprise customers can visit the following web page:

<https://securitybulletin.huawei.com/enterprise/en/security-advisory>

---

# Contents

---

<b>1 Edge Computing with IEF</b>	<b>1</b>
<b>2 Service Instances</b>	<b>3</b>
<b>3 User Guide (Professional)</b>	<b>7</b>
3.1 Node Management	7
3.1.1 Edge Node Overview	7
3.1.2 Configuring the Edge Node Environment	8
3.1.3 Registering an Edge Node	12
3.1.4 Managing an Edge Node	15
3.1.5 Upgrading an Edge Node	17
3.1.6 Logs, Monitoring, and Alarms	19
3.1.7 Installing and Configuring a GPU Driver	23
3.1.8 EdgeCore Configuration Management	25
3.1.9 Deleting an Edge Node	26
3.2 End Device Management	26
3.2.1 End Devices and Device Twins	26
3.2.2 Device Templates	28
3.2.3 End Devices	30
3.2.4 Binding an End Device to an Edge Node	30
3.2.5 Device Twin Working Principles	31
3.2.6 Migrating Device Data to the Cloud	34
3.2.7 Performing Security Authentication Using Certificate	41
3.2.8 MQTT Topics	50
3.2.8.1 Device Twin Update	50
3.2.8.2 Device Twin Delta	52
3.2.8.3 Device Member Update	53
3.2.8.4 Device Property Update	54
3.2.8.5 Device Member Acquisition	55
3.2.8.6 Device Member Acquisition Result	55
3.2.8.7 Device Twin Acquisition	56
3.2.8.8 Device Twin Acquisition Result	57
3.2.8.9 Device Twin Modification	58
3.2.8.10 Device Twin Modification Result	59

3.2.8.11 Encryption Data Request.....	60
3.2.8.12 Encryption Data Acquisition.....	61
3.2.8.13 Alarm Reporting.....	62
3.2.8.14 Alarm Clearance.....	67
3.2.8.15 Custom Topics.....	67
3.3 Containerized Application Management.....	68
3.3.1 Containerized Applications.....	68
3.3.2 Application Templates.....	77
3.3.3 ConfigMaps.....	82
3.3.4 Secrets.....	83
3.3.5 Encryption Data.....	85
3.3.6 Health Check Configuration.....	86
3.4 Edge-Cloud Messages.....	88
3.4.1 Edge-Cloud Message Overview.....	88
3.4.2 Cloud Delivering Messages to Edge Nodes.....	91
3.4.3 Edge Nodes Reporting Messages to the Cloud.....	95
3.4.4 System Subscriptions.....	98
3.5 Batch Management .....	102
3.5.1 Registering Nodes in Batches.....	102
3.5.2 Upgrading Nodes in Batches.....	107
3.5.3 Deploying Applications in Batches.....	109
3.5.4 Upgrading Applications in Batches.....	117
3.6 Auditing.....	118
3.6.1 IEF Operations Supported by CTS.....	118
3.6.2 Viewing Tracing Logs.....	121
3.7 Permissions Management.....	122
3.7.1 Creating a User and Granting Permissions.....	122
3.7.2 IEF Custom Policies.....	123
3.7.3 IEF Resources.....	124
3.7.4 IEF Request Conditions.....	124
3.7.5 Entrustment Description.....	127
<b>4 User Guide (Platinum).....</b>	<b>130</b>
4.1 Node Management.....	130
4.1.1 Edge Node Overview.....	130
4.1.2 Configuring the Edge Node Environment.....	131
4.1.3 Registering an Edge Node.....	135
4.1.4 Managing an Edge Node.....	139
4.1.5 Edge Node Groups.....	141
4.1.6 Upgrading an Edge Node.....	144
4.1.7 Logs, Monitoring, and Alarms.....	146
4.1.8 Installing and Configuring a GPU Driver.....	150
4.1.9 EdgeCore Configuration Management.....	152

4.1.10 Deleting an Edge Node.....	153
4.2 End Device Management.....	153
4.2.1 End Devices and Device Twins.....	153
4.2.2 Device Templates.....	155
4.2.3 End Devices.....	157
4.2.4 Binding an End Device to an Edge Node.....	157
4.2.5 Device Twin Working Principles.....	158
4.2.6 Migrating Device Data to the Cloud.....	161
4.2.7 Performing Security Authentication Using Certificate.....	168
4.2.8 MQTT Topics.....	177
4.2.8.1 Device Twin Update.....	177
4.2.8.2 Device Twin Delta.....	179
4.2.8.3 Device Member Update.....	180
4.2.8.4 Device Property Update.....	181
4.2.8.5 Device Member Acquisition.....	182
4.2.8.6 Device Member Acquisition Result.....	182
4.2.8.7 Device Twin Acquisition.....	183
4.2.8.8 Device Twin Acquisition Result.....	184
4.2.8.9 Device Twin Modification.....	185
4.2.8.10 Device Twin Modification Result.....	186
4.2.8.11 Encryption Data Request.....	187
4.2.8.12 Encryption Data Acquisition.....	188
4.2.8.13 Alarm Reporting.....	189
4.2.8.14 Alarm Clearance.....	194
4.2.8.15 Custom Topics.....	194
4.3 Containerized Application Management.....	195
4.3.1 Containerized Applications.....	195
4.3.2 Application Templates.....	205
4.3.3 Affinity and Anti-Affinity Scheduling.....	212
4.3.4 ConfigMaps.....	215
4.3.5 Secrets.....	216
4.3.6 Encryption Data.....	218
4.3.7 Health Check Configuration.....	220
4.4 Application Mesh.....	221
4.4.1 Scenario.....	221
4.4.2 Services.....	222
4.4.3 Gateways.....	225
4.5 Edge-Cloud Messages.....	228
4.5.1 Edge-Cloud Message Overview.....	228
4.5.2 Cloud Delivering Messages to Edge Nodes.....	231
4.5.3 Edge Nodes Reporting Messages to the Cloud.....	235
4.5.4 System Subscriptions.....	238

---

4.6 Batch Management .....	243
4.6.1 Registering Nodes in Batches.....	244
4.6.2 Upgrading Nodes in Batches.....	248
4.6.3 Deploying Applications in Batches.....	250
4.6.4 Upgrading Applications in Batches.....	257
4.7 Auditing.....	259
4.7.1 IEF Operations Supported by CTS.....	259
4.7.2 Viewing Tracing Logs.....	263
4.8 Permissions Management.....	263
4.8.1 Creating a User and Granting Permissions.....	263
4.8.2 IEF Custom Policies.....	265
4.8.3 Entrustment Description.....	266

# 1 Edge Computing with IEF

---

Intelligent EdgeFabric (IEF) provides you a complete edge computing solution where cloud applications are extended to the edge. By leveraging edge-cloud synergy, you can manage edge nodes and applications remotely while still processing data nearby. In addition, you can perform O&M in the cloud, including edge node monitoring, edge application monitoring, and log collection.

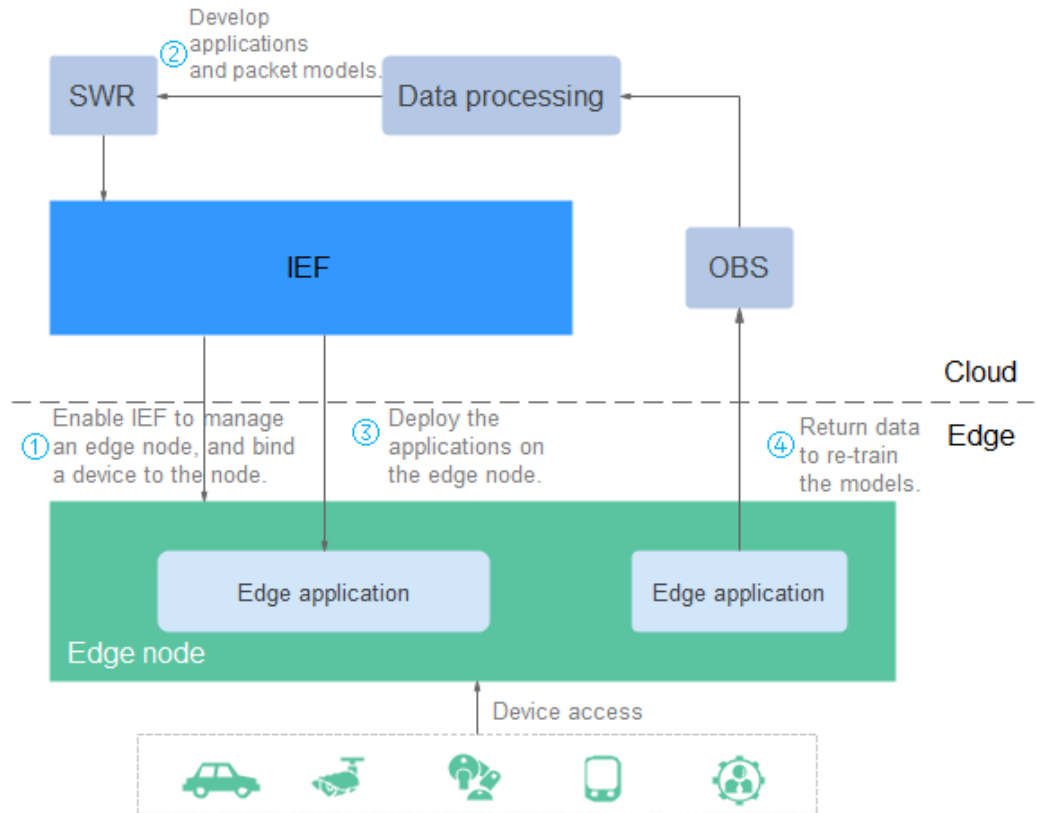
## Edge Computing with IEF

**Figure 1-1** shows how IEF implements edge computing.

1. Enable IEF to manage edge nodes, and bind end devices to the nodes.  
Install edge node software on edge nodes, and bind end devices to the nodes. Then you can use IEF to deploy applications on those edge nodes.  
For details on how to manage edge nodes and bind end devices, see [Node Management](#) and [End Device Management](#).
2. Develop applications, create images, and upload the images to [SoftWare Repository for Container \(SWR\)](#).  
This step ensures that edge nodes can pull images from SWR when IEF delivers applications.  
There is no sequential order between this step and [step 1](#). Either of them can be performed first as needed.
3. Deploy applications.  
After the edge nodes are managed and applications are developed, you can use IEF to deploy applications on edge nodes and start running your services. Running applications can be monitored and alarms can be reported through the Application Operations Management (AOM) service.  
For details on how to deploy applications, see [Containerized Application Management](#).
4. (Optional) Send the data back to the cloud for further processing, and update the applications based on the results.  
This step is not closely related to the use of IEF, but it is a common way to optimize applications based on processing data. You can determine whether to perform this step based on your requirements.



**Figure 1-1** Implementing edge computing



# 2 Service Instances

A service instance is a management cluster that IEF uses to manage edge nodes and deliver applications. IEF supports two types of service instances: Professional and Platinum.

- Professional (default): All users share the same management cluster. Professional service instances allow you to manage nodes, devices, and containerized applications.
- Platinum: Users need to create their own platinum service instances and then exclusively use those management clusters. Platinum service instances allow you to manage large-scale nodes and deliver higher performance. In addition to the functions provided by the professional service instance, functions such as batch job and application governance are supported by platinum service instances.

**Table 2-1** lists the differences between the two editions.

**Table 2-1** Functions provided by the two editions

Function	Description	Professional Edition	Platinum Edition
Edge node management	Registers and manages edge nodes.	√	√
End device management	Registers end devices and binds an end device to an edge node.	√	√
Containerized application management	Delivers containerized applications to an edge node.	√	√

Function	Description	Professional Edition	Platinum Edition
Edge-cloud message routing	Provides an edge-cloud message channel and supports edge-cloud message forwarding.	√	√
Multi-network access	Supports access to IEF through Internet, VPN, and Direct Connect.	√	√
Monitoring and O&M	Supports monitoring and O&M.	√	√
Batch job management	Creates and updates containerized applications in batches, and registers and updates edge nodes in batches.	√	√
Edge node group	Creates edge node groups. Multiple edge nodes with the same attributes (such as the hardware architecture) can form an edge node group for unified management.	×	√
Multi-instance	Supports multiple containerized application instances.	×	√
Exclusive clusters	Supports exclusive management plane clusters.	×	√
Application mesh	Supports service discovery and application traffic governance, such as load balancing.	×	√

Function	Description	Professional Edition	Platinum Edition
Add-ons	Supports add-on management.	√	√
Kubernetes native API openness	Supports operating Kubernetes clusters of a service instance through kubectl.	×	√

## Constraints

In-place capacity expansion of service instances is not supported. Therefore, edge nodes or applications that exceed the service instance quota can be managed only by creating service instances. Plan and create service instances of the proper scale in advance based on your service requirements.

## Creating a Platinum Service Instance

**Step 1** Log in to the IEF console.

**Step 2** In the navigation pane, choose **Dashboard**. On the page displayed on the right, click **Create Platinum Service Instance** in the upper right corner of the page.

**Step 3** Configure parameters.

- **Region:** Select the region where the service instance is deployed. Instances can only be used in the region specified when they are created. Select a region close to you for lower network latency and faster access.
- **Instance Name:** Enter the instance name.
- **Access Type:** **Internet** and **Direct Connect** are available. For details about how to connect edge nodes to IEF through Direct Connect, see [Connecting Edge Nodes to IEF Through Direct Connect or VPN](#).
- **Edge Nodes:** The number of edge nodes that can be managed by the service instance. Currently, 50, 200, and 1,000 nodes can be selected.
- **Access Bandwidth:** If **Access Type** is set to **Internet**, select **5 Mbit/s**, **10 Mbit/s**, or **30 Mbit/s** depending on the number of edge nodes. If **Access Type** is set to **Direct Connect**, the bandwidth is determined by the Direct Connect service.
- **Advanced Settings:** Multi-AZ deployment is supported. If this option is enabled, platinum service instances can be deployed in multiple AZs to facilitate disaster recovery. However, the cluster performance is decreased.

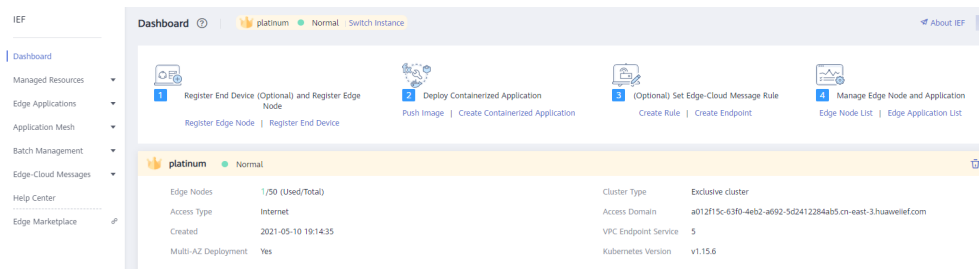
**Step 4** Click **Next**.

**Step 5** Confirm the order details, and click **Create**.

It takes 20–30 minutes to create a service instance. You can view the service instance status in the service instance list.

You can view the information about the platinum service instance on the **Dashboard** page.

**Figure 2-1** Viewing details of an instance



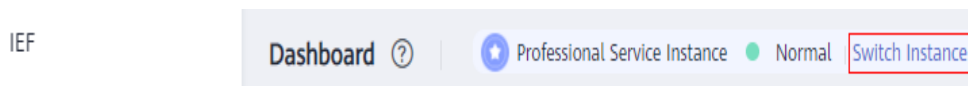
----End

## Switching an Instance

**Step 1** Log in to the IEF console.

**Step 2** In the navigation pane, choose **Dashboard**. On the page displayed on the right, click **Switch Instance** in the upper middle of the page.

**Figure 2-2** Switching an instance



**Step 3** Under the target instance, click **Select Instance**.

----End

# 3 User Guide (Professional)

---

## 3.1 Node Management

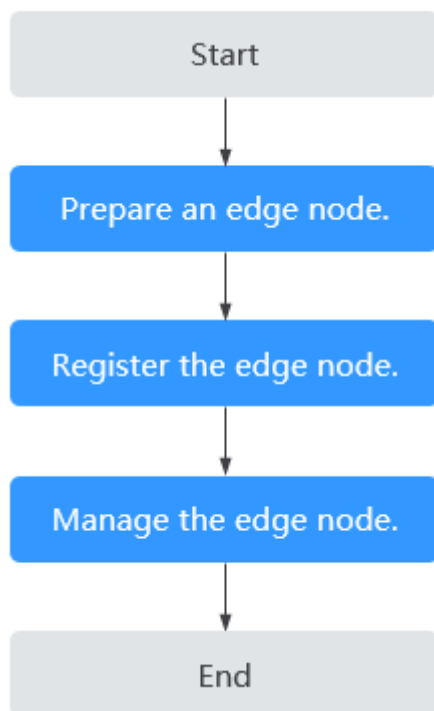
### 3.1.1 Edge Node Overview

Edge nodes are your own edge computing devices. They are used to run edge applications, process your data, and collaborate with cloud applications securely and conveniently. You can deploy system applications using IEF to extend cloud service capabilities to edge nodes, or deploy your own applications to implement edge computing capabilities.

To enable IEF to manage your edge node, perform the following steps:

1. Prepare the node. The edge node must meet requirements listed in [Configuring the Edge Node Environment](#).
2. Register the node on IEF, and download the EdgeCore installer and configuration file. For details, see [Registering an Edge Node](#).
3. Manage the edge node using the EdgeCore installer and configuration file. For details, see [Managing an Edge Node](#).

**Figure 3-1** Procedure



## 3.1.2 Configuring the Edge Node Environment

### Specifications Requirements

An edge node can be a physical machine or a virtual machine (VM). Edge nodes must meet the specifications listed in [Table 3-1](#).

**Table 3-1** Edge node requirements

Item	Specifications
OS	<p>The language of the operating system must be <b>English</b>.</p> <ul style="list-style-type: none"> <li>• x86_64 architecture Ubuntu LTS (Xenial Xerus), Ubuntu LTS (Bionic Beaver), CentOS, EulerOS, RHEL, Kylin, NewStart CGS Linux, NeoKylin, openEuler, Unity Operating System (UOS), Oracle Linux (OL), Huawei Cloud Euler (HCE), openEuler 23.09 Edge</li> <li>• Armv7i (Arm32) architecture Raspbian GNU/Linux (stretch)</li> <li>• AArch64 (Arm64) architecture Ubuntu LTS (Xenial Xerus), Ubuntu LTS (Bionic Beaver), CentOS, EulerOS, RHEL, Kylin, NewStart CGS Linux, NeoKylin, openEuler, Unity Operating System (UOS), Oracle Linux (OL), Huawei Cloud Euler (HCE), openEuler 23.09 Edge</li> </ul> <p><b>NOTE</b> The openEuler 23.09 Edge operating system is recommended for edge computing scenarios.</p>
Memory	<p>More than 256 MB of memory is recommended as 128 MB of memory is required to run the edge software.</p>
CPU	<p>≥ 1 core</p>
Hard disk	<p>≥ 1 GB</p>
GPU (optional)	<p>The GPU models on the same edge node must be the same.</p> <p><b>NOTE</b> Currently, NVIDIA Tesla GPUs such as P4, P40, and T4 are supported.</p> <p>If an edge node is equipped with GPUs, you can choose not to enable its GPUs when registering it on IEF.</p> <p>If you choose to enable GPUs of an edge node, the GPU driver has to be installed on the edge node before you can manage it on IEF.</p> <p>Currently, only x86-based GPU nodes can be managed by IEF.</p>



Item	Specifications
NPU (optional)	<p>Ascend AI processors</p> <p><b>NOTE</b> Currently, edge nodes integrated with Ascend Processors are supported, such as Atlas 300 inference cards, and Atlas 800 inference servers. Supported NPU specifications include Ascend 310P, 310B, Ascend 310P-share, and virtualization partition NPUs..</p> <p>If you choose to enable NPUs of an edge node, ensure that the NPU driver has been installed on it. Currently, Ascend 310 supports only firmware versions 1.3.x.x and 1.32.x.x, for example, 1.3.2.B893. You can run the <b>npu-smi info</b> command to view your firmware version.The NPU driver version must be 22.0.4 or later. You can go to the driver path, for example, <b>/usr/local/Ascend/driver</b>, and run the <b>cat version.info</b> command to view your driver version. If the driver is not installed, contact the device manufacturer for assistance.</p>
Container engine	<p>The Docker version must be later than 17.06. If Docker 1.23 or later is used, set the docker cgroupfs version to 1. Docker HTTP API v2 is not supported.</p> <p>(However, Docker 18.09.0 is not recommended as it has a serious bug. For details, see <a href="https://github.com/docker/for-linux/issues/543">https://github.com/docker/for-linux/issues/543</a>. If this version has been installed, upgrade it at the earliest possible opportunity. )</p> <p><b>NOTICE</b> After Docker is installed, configure the Docker process to start at host startup. This configuration prevents system exceptions caused by Docker startup failures after the host is restarted.</p> <p>Docker <b>Cgroup Driver</b> must be set to <b>cgroupfs</b>. For details, see <a href="#">How Do I Set Docker Cgroup Driver After Installing Docker on an Edge Node?</a>.</p>
Glibc	The Glibc version must be later than 2.17.
Port	Edge nodes require port 8883, which is the listening port of the built-in MQTT broker on edge nodes. Ensure that this port works properly.
Time synchronization	The time on an edge node must be consistent with the UTC time. Otherwise, the monitoring data and logs of the edge node may be inaccurate. You can select an NTP server for time synchronization. For details, see <a href="#">How Do I Synchronize Time with the NTP Server?</a>

## Configuring the Edge Node Environment

**Step 1** Log in to an edge node as a user with sudo permissions.

**Step 2** Configure the GPU driver.

If your edge node has been equipped with a GPU, install and configure the GPU driver on the edge node. For details, see [Installing and Configuring a GPU Driver](#).

**Step 3** Configure the NPU driver.

If your edge node uses Ascend AI processors, ensure that the corresponding driver has been installed.

**Step 4** Install Docker on the edge node and check the Docker status.

The Docker version must be later than 17.06. Docker 18.06.3 is recommended. However, Docker 18.09.0 is not recommended as it has a serious bug. If you have used this version, upgrade it at the earliest possible opportunity.

After Docker installation is complete, run **docker -v** to check whether Docker was installed properly. If the following information is displayed, Docker was installed properly.

```
# docker -v
Docker version 19.03.12, build 48a66213fee
```

**Step 5** Configure firewall rules for the edge node.

Check the firewall status on the edge node.

```
systemctl status firewalld
firewall-cmd --state
```

In the command output, **not running** indicates that the firewall is disabled and **running** indicates that the firewall is enabled.

If the firewall is enabled, enable port 8883 or disable the firewall.

- To enable port 8883, run the following commands:

```
firewall-cmd --add-port=8883/tcp --permanent
systemctl restart firewalld
```

- To disable the firewall, run the following commands:

```
systemctl disable firewalld
systemctl stop firewalld
```

----End

## Security Tips

To improve host security, you are advised to harden the OS of the edge node by performing the following operations:

1. Set strong passwords for all OS accounts (including administrators and common users), database accounts, and application (web) system management accounts. Each password must contain at least 12 characters.
2. Do not run applications using the administrator account. Disallow applications (such as webs) to use the database administrator account to interact with databases. Configure security groups and open only necessary ports to the public network. Protect the service web console ports and LAN internal communication ports from being exposed to the public network. Disable high-risk ports (such as the SSH port), allow limited source IP addresses to access the ports, or use the O&M channel established based on VPNs or bastion hosts.

3. Periodically back up service data remotely to prevent data loss caused by intrusions.
4. Periodically detect security vulnerabilities in the system and software, update system security patches in a timely manner, and upgrade the software to the latest official version.
5. Download and install the software from official channels. For the software downloaded from non-official channels, use antivirus software to scan it before running.

If you use Huawei Cloud Elastic Cloud Server (ECS), perform the following operations:

1. Set the host login mode to key login.
2. Use Huawei Cloud [Host Security Service \(HSS\)](#) for in-depth defense.

### 3.1.3 Registering an Edge Node

Registering an edge node is the process of creating an edge node on IEF and obtaining the node configuration file and EdgeCore Installer.

#### Constraints

- NPUs on the same node support only the same partition specification.
- You cannot perform virtualization partition on the NPU of an IEF-managed edge node by simply upgrading the NPU plug-in to v2.1.0. Uninstall and reinstall the node, manually partition the NPU, and then manage the node on IEF again.
- The NPU driver version must be later than 22.0. You can go to the driver path (for example, `/usr/local/Ascend/driver`) and run the `cat version.info` command to view your driver version.
- If you enable the NPU sharing mode, you need to set the container sharing mode for all chips on the NPU.
- NPU virtualization partition is not supported if the D310P-share is used.

#### Registering an Edge Node

**Step 1** Log in to the IEF console.

**Step 2** In the navigation pane, choose **Managed Resources > Edge Nodes**. Then, click **Register Edge Node** in the upper right corner.

**Step 3** Configure basic edge node information.

**Figure 3-2** Basic edge node information (1)

The screenshot displays the configuration interface for an edge node. It includes the following elements:

- Service Instance:** A radio button selection for 'Professional Service Instance'.
- Node Name:** A text input field with the placeholder 'Enter a node name.'
- Description:** A larger text area with the placeholder 'Enter an edge node description.' and a character count '0/255'.
- Tags:** Two input fields for 'Tag key' and 'Tag value', with a note 'You can add 20 more tags.'
- AI Accelerator Card:** Three buttons: 'Not installed' (selected), 'Ascend AI accelerator card', and 'NVIDIA GPU'.
- Bind Device:** A table with columns 'Device Names', 'Device to Node Relationship', 'Remarks', and 'Operation'. Below the table is an 'Add Device' button.
- Docker:** A dropdown menu set to 'Enable'.
- Listening Address:** Two buttons: 'NIC' (selected) and 'IP'. Below them is a list of listening addresses:
  - tls:// lo : 8883 (with a delete icon)
  - tls:// docker0 : 8883 (with a delete icon)
 An 'Add Listening Address' button is located at the bottom of the list.

- **Node Name:** name of an edge node. The name can contain 1 to 64 characters, including letters, digits, hyphens (-), and underscores (\_).
- **Description:** Optional. Enter the description of the edge node.
- **Tags:** Tags can be used to classify resources, facilitating resource management.
- **AI Accelerator Card**

Ascend AI accelerator card: supports edge nodes that support Ascend processors. If you need to use Ascend 310, 310B, first select the **AI accelerator card** and then select the NPU type. [Table 3-2](#) lists the NPU types supported by Ascend AI accelerator cards.

**Table 3-2** NPU types

Type	Description
Ascend 310	Ascend 310 chips
Ascend 310B	Ascend 310B chips

**NVIDIA GPU:** Select this option if your edge node is equipped with an NVIDIA GPU.

**Not installed:** Select this option if your edge node does not use any AI accelerator card.

 **NOTE**

If you select **NVIDIA GPU** but your edge node is not equipped with any NVIDIA GPU, the edge node cannot be managed by IEF.

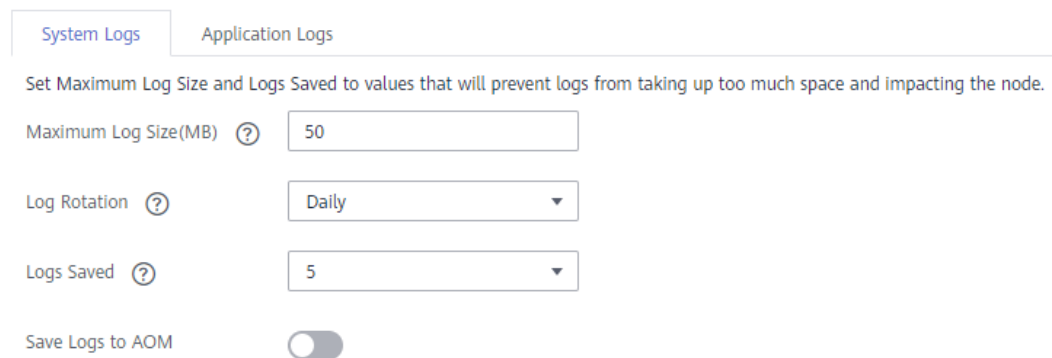
If your edge node has been equipped with a GPU, install and configure the GPU driver on the edge node before you can manage it on IEF. For details, see [Installing and Configuring a GPU Driver](#).

- **Bind Device:** Bind an end device to the edge node. If you have not registered an end device, register one by referring to [End Device Management](#). If you do not bind an end device in this step, you can bind one after the edge node is registered.
- **Docker:** After Docker is enabled, containerized applications can be deployed.
- **Listening Address**

The address on which the built-in MQTT broker of the edge node listens. It is used for edge-cloud messaging. For details about edge-cloud messaging, see [Edge-Cloud Message Overview](#).

By default, local NICs lo (localhost) and docker0 are listened to. You can specify an NIC name or IP address to set the NIC to be listened to. Multiple NICs or IP addresses can be added.

**Figure 3-3** Basic edge node information (2)



The screenshot shows a configuration interface for edge node logs. It has two tabs: 'System Logs' (selected) and 'Application Logs'. Below the tabs is a warning: 'Set Maximum Log Size and Logs Saved to values that will prevent logs from taking up too much space and impacting the node.' There are three input fields: 'Maximum Log Size (MB)' with a value of 50, 'Log Rotation' with a dropdown set to 'Daily', and 'Logs Saved' with a dropdown set to 5. At the bottom, there is a toggle switch for 'Save Logs to AOM' which is currently turned off.

Currently, system and application logs can be collected from edge nodes.

- **System Logs:** logs generated by the IEF software (such as edge-core, edge-logger, and edge-monitor) installed on edge nodes.
- **Application Logs:** logs generated by applications deployed on edge nodes.

Set the following parameters for both system and application logs:

- **Maximum Log Size (MB):** maximum size of a log file, in MB. The default value is **50**. The value range is 10–1000. If a log file reaches the size specified here, the system compresses the log file and dumps it to a specified directory on the host.
  - System logs are stored in the **/var/IEF/sys/log/** directory on edge nodes and will be dumped to AOM when the maximum log size is reached.
  - Application logs include the standard output of containers and the logs mounted to **/var/IEF/app/log** on the edge node. These logs will be dumped to AOM when the maximum log size is reached.

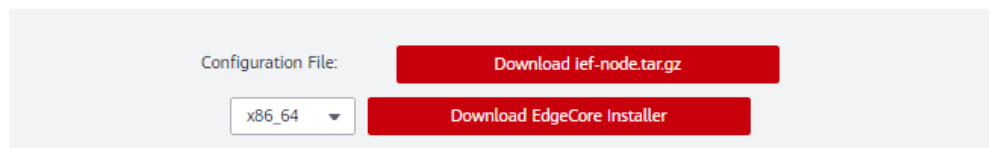
- **Log Rotation:** log dump interval. You can select **Daily, Weekly, Monthly, or Yearly**. Log files will be dumped when either **Maximum Log Size (MB)** or **Log Rotation** is reached.
- **Logs Saved:** maximum number of logs being retained at a specified period. The default value is **5**. The value range is 1–10. Once this number is reached, when new log files are dumped in, old log files will be deleted on a first-in first-out (FIFO) basis.
- **Save Logs to AOM**  
You can enable or disable log uploading to AOM. If it is enabled, you can view the logs in AOM. For details, see [Viewing Logs on AOM](#).

**Step 4** Select **I have read and agree to the Huawei Cloud Service Level Agreement**. Click **Register** in the lower right corner. You can choose to register the edge node using a certificate or a token.

**Registration using a certificate:** Download the configuration file and EdgeCore Installer, which will be used in [Managing an Edge Node](#).

**Figure 3-4** Downloading the configuration file and EdgeCore Installer

Download the following configuration files and software to complete the creation.  
Download the certificate right now because you will not be able to retrieve it afterwards.



1. Click **Download *Edge node name.tar.gz***.
2. Select the CPU architecture of your edge node, and click **Download EdgeCore Installer**.

**Step 5** In the lower right corner, select **I've finished downloading** and click **Finish**.

The edge node is in the **Unconnected** state because the EdgeCore Installer downloaded in [Registering an Edge Node](#) has not been installed. For details, see [Managing an Edge Node](#).

**Figure 3-5** Unconnected edge node

Name/ID	Node Status	Host Name/Network	Instances (Normal/All)	Created	Edge Software...	Node Tags	Node Source	Operation
ief-node 569d31f4-83de-4699-abc	Unconnected Installation Guide		0/0	Jul 20, 2021 09:20:23 GMT...			Custom node	Delete   More

----End

## Follow-up Procedure

After the registration is complete, you need to manage the edge node. For details, see [Managing an Edge Node](#).

### 3.1.4 Managing an Edge Node

Managing an edge node is the process of using the EdgeCore installer and configuration file downloaded in [Registering an Edge Node](#) to install EdgeCore

on the edge node. In this way, the edge node can be connected to and managed by IEF.

When you manage your edge node on IEF for the first time, IEF automatically pushes the latest EdgeCore. For example, if three EdgeCore versions 2.51.0, 2.52.0, and 2.53.0 are available, IEF will push the latest version 2.53.0.

#### NOTE

The edge nodes registered on IEF have one-to-one relationships with the actual devices. The EdgeCore installer and configuration file of an edge node can be installed on only one actual device.

## Prerequisites

- You have prepared a node that meets the specified environment requirements. For details, see [Configuring the Edge Node Environment](#).
- You have registered a node on IEF and obtained the node configuration file and EdgeCore installer. For details, see [Registering an Edge Node](#).

## Managing an Edge Node

**Step 1** Log in to an edge node as a user with sudo permissions.

**Step 2** Upload the EdgeCore installer and configuration file downloaded in [Registering an Edge Node](#) to a specified directory on the edge node, for example, `/home`, and go to the directory.

**Step 3** Run the following command to decompress the EdgeCore installer to the `/opt` directory:

```
sudo tar -zxvf edge-installer_1.0.0_x86_64.tar.gz -C /opt
```

Replace `edge-installer_1.0.0_x86_64.tar.gz` with the name of the EdgeCore installer package downloaded in [Registering an Edge Node](#).

**Step 4** Run the following command to decompress the configuration file to the `opt/IEF/Cert` directory. If the edge node to be managed is registered using a token, skip this step.

```
sudo mkdir -p /opt/IEF/Cert; sudo tar -zxvf Edge node name.tar.gz -C /opt/IEF/Cert
```

Replace `Edge node name.tar.gz` with the name of the configuration file downloaded in [Registering an Edge Node](#).

**Step 5** Run one of the following commands to manage the edge node:

- For an edge node registered using a certificate:

```
cd /opt/edge-installer; sudo ./installer -op=install
```

- For an edge node registered using a token:

```
cd /opt/edge-installer; sudo ./installer -op=install -identifier=Credential for registration using a token
```

Replace `Credential for registration using a token` with the value of `identifier` field of the installation credential saved in [Registering an Edge Node](#).

**Step 6** Verify whether the edge node is managed successfully.

1. Log in to the IEF console.
2. In the navigation pane, choose **Managed Resources > Edge Nodes**.
3. Check the edge node status. If the status is **Running**, the edge node has been managed by IEF.

**Figure 3-6** Checking the edge node status

<input type="checkbox"/>	Name/ID	Node Status	Host Name/Network	Instances (Normal/A...	Created	Edge Software Version
<input type="checkbox"/>	ief-node 7092ad14-adee-4a09-b969-1505	Running	eth0:192.168.0.230	0/0	Jul 20, 2021 09:3...	2.52.0 <a href="#">Upgradeable</a>

----End

**NOTICE**

After the edge node is managed by IEF, do not delete the `/opt` directory on the edge node, or you will need to register the edge node and enable IEF to manage it again.

## 3.1.5 Upgrading an Edge Node

### Context

IEF releases new versions of EdgeCore irregularly. You can upgrade the EdgeCore software installed on your edge node as required.

### Version Support Policy

IEF only maintains the edge node software versions released within one year. Therefore, you are advised to upgrade your edge nodes at least once a year.

### Version Upgrade Rule

During edge node upgrade, IEF automatically selects the EdgeCore of the latest version.

For example, if EdgeCore 2.22.0, 2.23.0, and 2.24.0 are available and the EdgeCore version on your edge node is 2.12.0, IEF will push the EdgeCore 2.24.0 to your edge node.

### Precautions

- IEF does not proactively upgrade EdgeCore on your edge nodes. You are advised to upgrade the node manually in the time window with the minimum impact on your services.
- Upgrading a version within the maintenance period will not interrupt your applications running on the edge node. However, if message routing is used, services may be temporarily affected.
- Upgrading a version beyond the maintenance period may temporarily interrupt services due to container restart.



- Do not change node configurations during the node upgrade, such as restarting Docker, installing or uninstalling the GPU/NPU driver, upgrading the OS kernel, or modifying network configurations, which may result in node upgrade failures.

## Procedure

**Step 1** Log in to an edge node, and configure firewall rules.

Check the firewall status on the edge node.

```
systemctl status firewalld
firewall-cmd --state
```

In the command output, **not running** indicates that the firewall is disabled and **running** indicates that the firewall is enabled.

If the firewall is enabled, enable port 8883 or disable the firewall.

- To enable port 8883, run the following commands:  

```
firewall-cmd --add-port=8883/tcp --permanent
systemctl restart firewalld
```
- To disable the firewall, run the following commands:  

```
systemctl disable firewalld
systemctl stop firewalld
```

**Step 2** Log in to the IEF console.

**Step 3** In the navigation pane, choose **Managed Resources > Edge Nodes**.

**Step 4** View the **Edge Software Version** column to check whether an upgrade can be performed.

Only edge nodes in the **Running** state can be upgraded.

- If **Upgradeable** is displayed in the **Edge Software Version** column, you can perform an upgrade.
- If **Upgradeable** is not displayed, check whether the edge node is in the **Running** state. If the edge node is in the **Running** state and **Upgradeable** is not displayed, the edge node already runs the latest EdgeCore.

**Figure 3-7** Checking whether an edge node can be upgraded

Name/ID	Node	Host Name/Netw...	Instances (No...	Created	Edge Soft...	Node Tags	Node S...	Operation
ief-node 7092ad14-adee-4a09-b969-1505bbdec	Running	eth0:192.168.0.230	0/0	Jul 20, 2021 09:33:26 GMT...	2.52.0 Upgradeable	--	Custom...	Delete   More

**Step 5** Choose **More > Upgrade**.

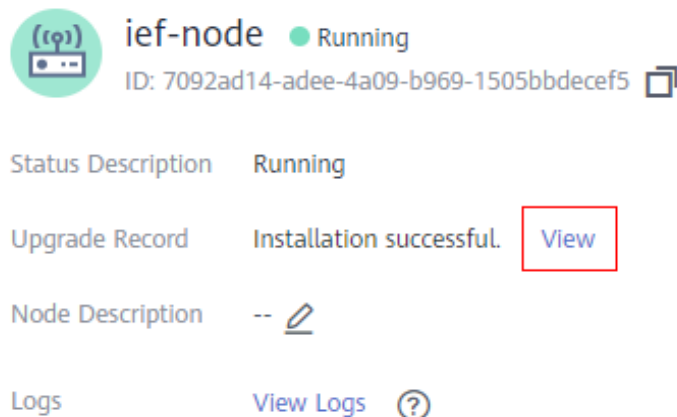
**Figure 3-8** Upgrading an edge node

Name/ID	Node	Host Name/Netw...	Instances (No...	Created	Edge Soft...	Node Tags	Node S...	Operation
ief-node 7092ad14-adee-4a09-b969-1505bbdec	Running	eth0:192.168.0.230	0/0	Jul 20, 2021 09:33:26 GMT...	2.52.0 Upgradeable	--	Custom...	Delete   More

Enable  
 Disable  
 View Monitoring  
 Upgrade

**Step 6** Click the node name to go to the node details page, and view the upgrade record.

Figure 3-9 Upgrade record



----End

## 3.1.6 Logs, Monitoring, and Alarms

### Log Description

If the log function is enabled for an edge node on the IEF console, the system and application logs generated for the edge node will be uploaded to AOM.

- **System Logs:** logs generated by the IEF software (such as edge-core, edge-logger, and edge-monitor) installed on edge nodes.
- **Application Logs:** logs generated by applications deployed on edge nodes.
  - Edge nodes will upload logs in the `/var/IEF/app/log` directory to AOM through IEF. You can mount the `/var/IEF/app/log/{appName}` directory into the container when creating an application. For details, see [hostPath: used for mounting a directory of the host into the container](#). You can view logs on AOM by `{appName}`.
  - Edge nodes will also upload container logs in `{{DOCKER_ROOT_DIR}}/containers/{containerID}/{containerID}-json.log` to AOM. You can run the `docker info` command to query the value of `DOCKER_ROOT_DIR`. `containerID` indicates the container ID.

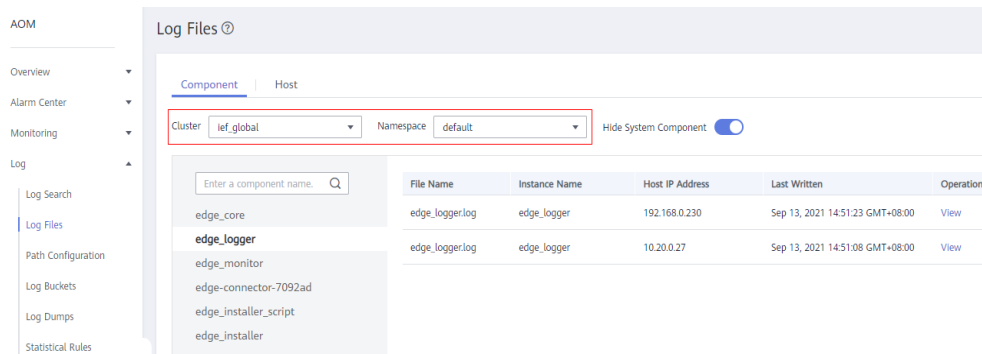
### Viewing Logs on AOM

**Step 1** Log in to the AOM console.

**Step 2** In the navigation pane, choose **Log > Log Files**, and click the **Component** tab.

**Step 3** On the displayed page, select cluster **ief\_global** and namespace **default**.

**Figure 3-10** Selecting a cluster and a namespace



**Step 4** Search for logs by application name, and click **View** in the row where the log file resides to view detailed logs.

----End

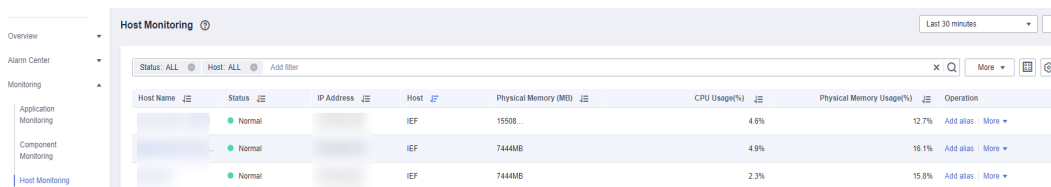
## Viewing Node Monitoring Information on AOM

You can view the node monitoring information on AOM.

**Step 1** Log in to the AOM console.

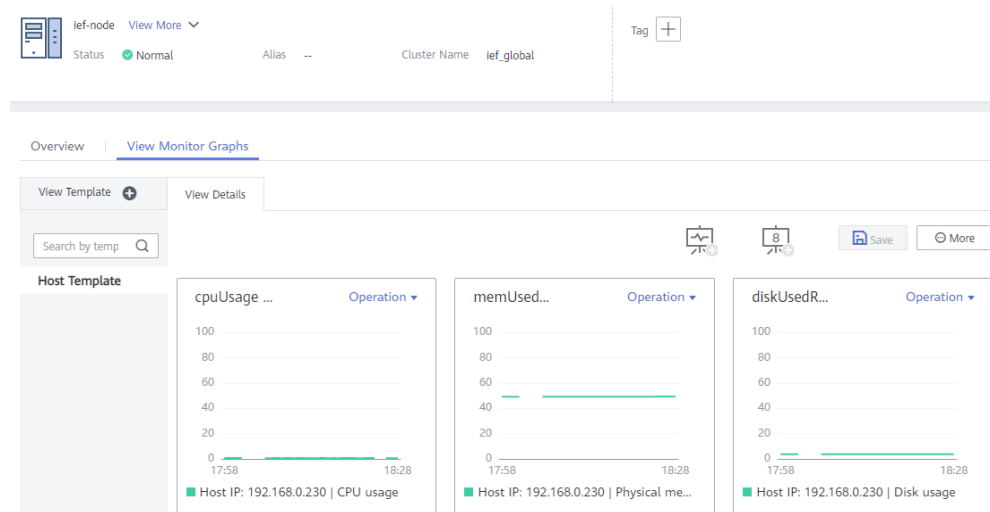
**Step 2** Click the name of the node whose monitoring information is to be viewed.

**Figure 3-11** Selecting a monitored node



**Step 3** On the **View Monitor Graphs** tab page, view the resource usage of the node, such as the CPU usage and memory usage.

**Figure 3-12** Viewing monitoring information



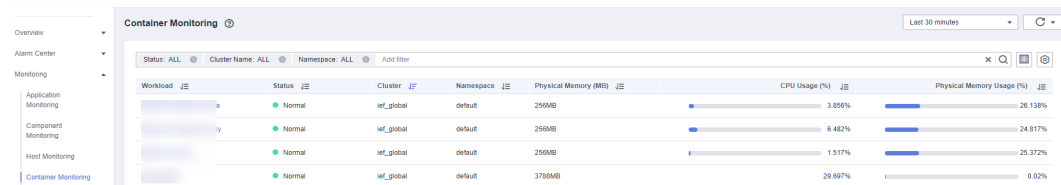
----End

## Viewing Container Monitoring Information on AOM

You can view the monitoring information about the containerized applications deployed on edge nodes on AOM.

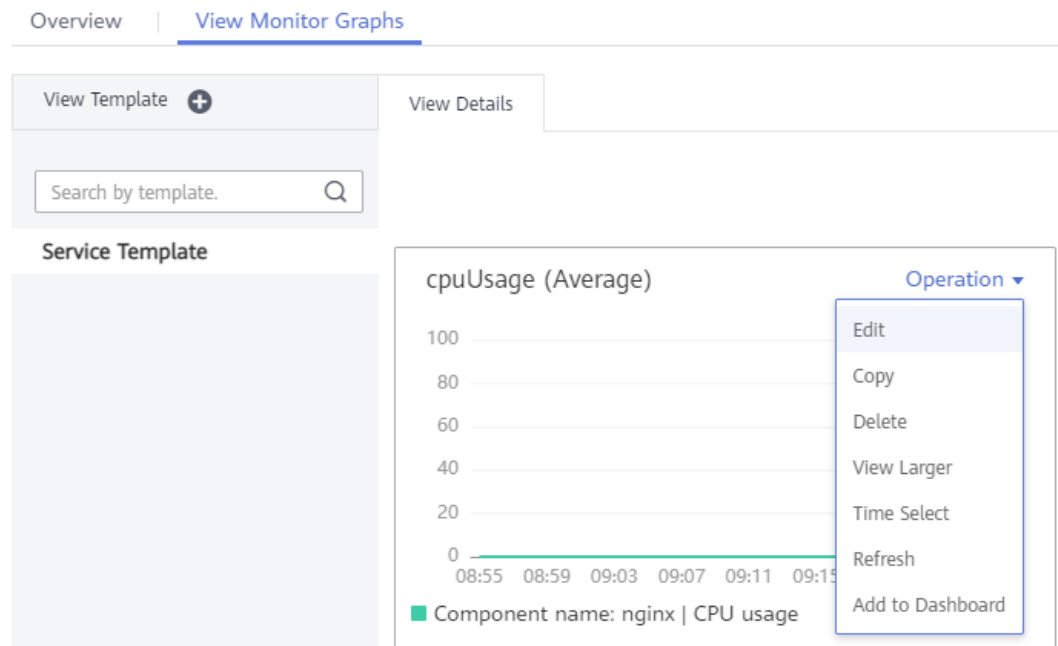
- Step 1** Log in to the AOM console.
- Step 2** Select the workload whose monitoring information is to be viewed.

**Figure 3-13** Selecting a workload



- Step 3** On the **View Monitor Graphs** tab page, view the monitoring metrics of the container, such as the CPU usage and memory usage.

**Figure 3-14** Viewing monitoring information



----End

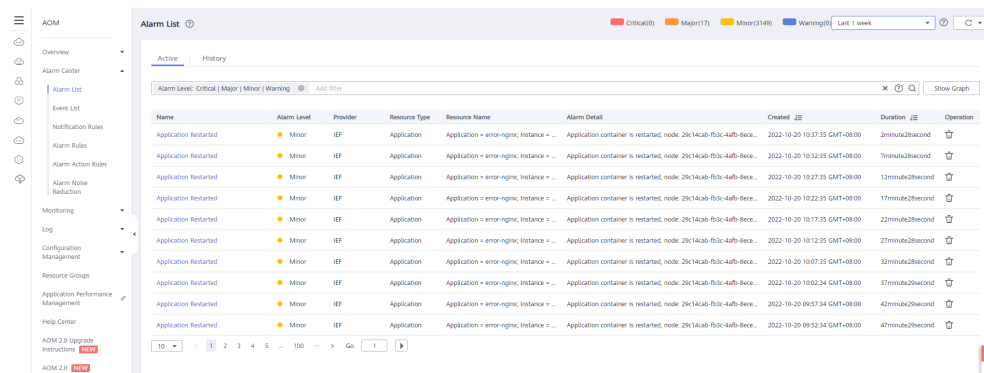
## Alarms Preset on IEF

IEF comes with seven types of alarms preconfigured for each edge node. These alarms will be automatically reported to AOM.

Alarm Name	Trigger	Clearance Condition	Severity
Container Engine Is Abnormal	Docker has been enabled on an edge node, and the Docker information fails to be queried.	Docker is running properly, and EdgeCore can obtain the Docker information.	Critical
Application Liveness Probe Is Abnormal	A liveness probe has been configured for the application, and the probe detects an exception.	The probe detects that the container is normal.	Major
Failed to Obtain GPU Resources	GPU resources could not be obtained during GPU application deployment.	GPU resources are obtained.	Critical
Failed to Obtain GPU Information	GPU has been enabled on an edge node, and the GPU information fails to be queried.	The GPU information is successfully queried.	Critical

Alarm Name	Trigger	Clearance Condition	Severity
Invalid AK/SK	EdgeHub has distributed 10 consecutive temporary AK/SK pairs and detects that the AK/SK has expired or is in abnormal state.	EdgeHub successfully distributes the temporary AK/SK.	Major
Application Restarted	The application container restarts unexpectedly.	This alarm does not need to be cleared.	Minor
NIC Bound to the Container Is Faulty	The NIC bound to the container is faulty.	The NIC bound to the container becomes normal.	Critical

Figure 3-15 Viewing alarms



## Setting Alarms on AOM

You can create alarm rules on AOM to monitor metrics of edge nodes. For details, see [Creating a Threshold Rule](#).

## Reporting User-Defined Alarms to AOM

IEF can report customized alarms from edge nodes to AOM. To be specific, after the MQTT client publishes alarm information to the MQTT broker, IEF will automatically report the alarms to AOM.

For details, see [Alarm Reporting](#) and [Alarm Clearance](#).

## 3.1.7 Installing and Configuring a GPU Driver

### Context

For an edge node that uses GPUs, you need to install and configure the GPU driver before managing the edge node on IEF.

Currently, IEF supports NVIDIA Tesla GPUs such as P4, P40 and T4, and the GPU drivers that match CUDA Toolkit 8.0 to 10.0.

## Procedure

**Step 1** Install the GPU driver.

1. Download the GPU driver. The recommended driver link is as follows:  
[https://www.nvidia.com/content/DriverDownload-March2009/confirmation.php?url=/tesla/440.33.01/NVIDIA-Linux-x86\\_64-440.33.01.run&lang=us&type=Tesla](https://www.nvidia.com/content/DriverDownload-March2009/confirmation.php?url=/tesla/440.33.01/NVIDIA-Linux-x86_64-440.33.01.run&lang=us&type=Tesla)
2. Run the following command to install the GPU driver:  
**bash NVIDIA-Linux-x86\_64-440.33.01.run**
3. Run the following command to check the GPU driver installation status:  
**nvidia-smi**

**Step 2** Log in to the edge node as user **root**.

**Step 3** Run the following command:

```
nvidia-modprobe -c0 -u
```

**Step 4** Create directories.

```
mkdir -p /var/IEF/nvidia/drivers /var/IEF/nvidia/bin /var/IEF/nvidia/lib64
```

**Step 5** Copy GPU driver files to the directories.

- For CentOS, run the following commands in sequence to copy the driver files:  

```
cp /lib/modules/{Kernel version of the current environment}/kernel/drivers/video/nvi* /var/IEF/nvidia/drivers/  
cp /usr/bin/nvidia-* /var/IEF/nvidia/bin/  
cp -rd /usr/lib64/libcuda* /var/IEF/nvidia/lib64/  
cp -rd /usr/lib64/libEG* /var/IEF/nvidia/lib64/  
cp -rd /usr/lib64/libGL* /var/IEF/nvidia/lib64/  
cp -rd /usr/lib64/libnv* /var/IEF/nvidia/lib64/  
cp -rd /usr/lib64/libOpen* /var/IEF/nvidia/lib64/  
cp -rd /usr/lib64/libvdpau_nvidia* /var/IEF/nvidia/lib64/  
cp -rd /usr/lib64/vdpau /var/IEF/nvidia/lib64/
```
- For Ubuntu, run the following commands in sequence to copy the driver files:  

```
cp /lib/modules/{Kernel version of the current environment}/kernel/drivers/video/nvi* /var/IEF/nvidia/drivers/  
cp /usr/bin/nvidia-* /var/IEF/nvidia/bin/  
cp -rd /usr/lib/x86_64-linux-gnu/libcuda* /var/IEF/nvidia/lib64/  
cp -rd /usr/lib/x86_64-linux-gnu/libEG* /var/IEF/nvidia/lib64/  
cp -rd /usr/lib/x86_64-linux-gnu/libGL* /var/IEF/nvidia/lib64/  
cp -rd /usr/lib/x86_64-linux-gnu/libnv* /var/IEF/nvidia/lib64/  
cp -rd /usr/lib/x86_64-linux-gnu/libOpen* /var/IEF/nvidia/lib64/  
cp -rd /usr/lib/x86_64-linux-gnu/libvdpau_nvidia* /var/IEF/nvidia/lib64/  
cp -rd /usr/lib/x86_64-linux-gnu/vdpau /var/IEF/nvidia/lib64/
```

You can run the **uname -r** command to view the kernel version of the current environment. The following is an example. Replace the kernel version with the actual value.

```
# uname -r  
3.10.0-514.e17.x86_64
```

**Step 6** Run the following command to change the directory permissions:

```
chmod -R 755 /var/IEF
```

```
----End
```

## 3.1.8 EdgeCore Configuration Management

### Scenario

IEF allows you to manage EdgeCore configuration parameters, through which you can let EdgeCore work under your very requirements.

### Procedure

**Step 1** Run the following command on the edge node to modify the EdgeCore configuration and save the modification:

```
vi /opt/IEF/Edge-core/conf/edge.yaml
```

The following table describes the parameters that can be configured.

**Table 3-3** Parameter description

Component	Parameter	Description	Value
edge-core	interface-name	NIC name	Default value: <b>eth0</b>
	internal-server	Listening address of the built-in MQTT broker	tls://lo:8883,tls://docker0:8883
	image-gc-high-threshold	Percentage of the disk usage that triggers image garbage collection	Default value: <b>80</b>
	image-gc-low-threshold	Target percentage of the disk usage after releasing resources with image garbage collection	Default value: <b>40</b>
	swr-url	Proxy address for pulling an image	Default value: ""

**Step 2** After the configuration is modified, restart EdgeCore.



```
systemctl restart edgecore
```

```
----End
```

## 3.1.9 Deleting an Edge Node

### Prerequisites

Before deleting an edge node, you need to unbind the end devices and delete applications and certificates on the edge node.

### Procedure

- Step 1** Run the **sudo** command to log in to the edge node.
- Step 2** Run the following commands to uninstall the software and configuration files on the managed nodes:  

```
cd /opt/edge-installer; sudo ./installer -op=uninstall
```
- Step 3** Log in to the IEF console.
- Step 4** In the navigation pane, choose **Managed Resources > Edge Nodes**.
- Step 5** Locate the row which contains the edge node to be deleted. In the **Operation** column, choose **More > Delete**.
- Step 6** Delete the node as prompted.

```
----End
```

## 3.2 End Device Management

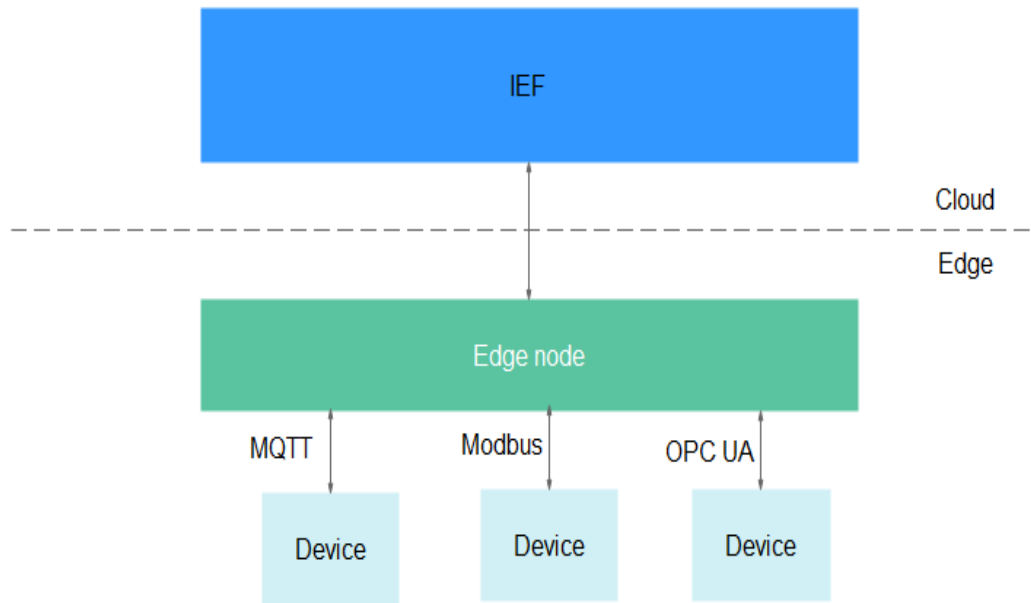
### 3.2.1 End Devices and Device Twins

#### End Device

End devices can be as small as a sensor or controller or as large as a smart camera or computer numerical control (CNC) machine tool.

These devices can be connected to IEF through edge nodes by using MQTT. After end devices are connected to IEF, you can manage them on IEF in a unified manner.

**Figure 3-16** End device management

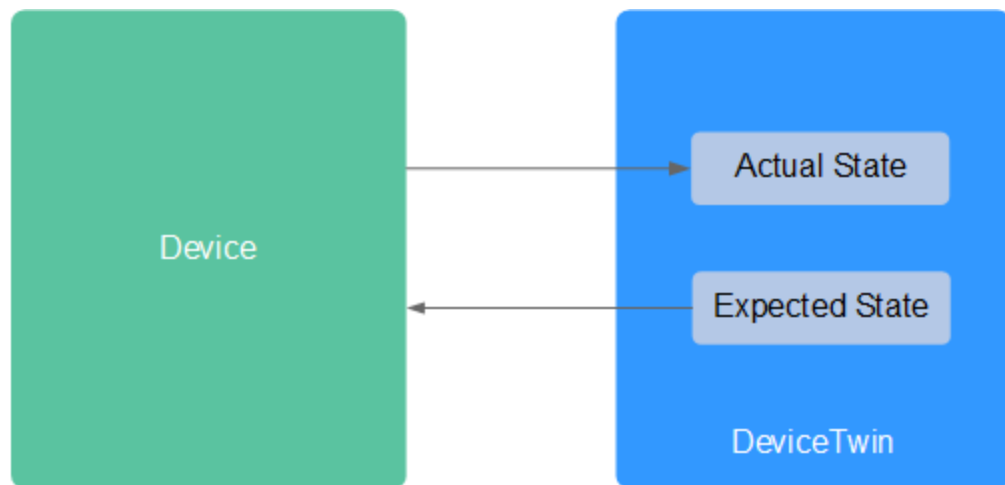


## Device Twins

An end device usually has two types of data:

- Metadata that does not change, which includes the serial number, asset identifier, MAC address, and other information about the device. This type of data can be called static properties or device properties.
- Dynamic device data, which includes dedicated real-time device data for a specific scenario, for example, the on and off states of a light. This type of data can be called twin properties.

DeviceTwin has the same features as physical devices and facilitates better communication between end devices and applications. The commands sent by an application reach DeviceTwin (an internal component on an edge node). Then, DeviceTwin updates the device status according to **Expected State** set by the application. In addition, the end device reports its **Actual State** in real time. DeviceTwin records both **Actual State** and **Expected State** of the end device. In this way, the status of an offline device can be synchronized when it comes back online.

**Figure 3-17** DeviceTwin

On the IEF console, you can register an end device and bind it to an edge node. After the binding, the device and twin properties will be stored on the edge node. Applications deployed on the edge node can obtain device and twin properties, and modify **Expected State** and **Actual State** recorded in DeviceTwin. In addition, IEF synchronizes twin properties between the cloud and edge. If a conflict occurs, the properties at the edge are used.

For details about the edge-cloud synergy for the end device status, see [Device Twin Working Principles](#).

## Procedure

To enable IEF to manage and control an end device, perform the following steps:

1. Define a device template (including device and twin properties).
2. Use the template to register a device.  
You can also register a device without using a template.
3. Bind the device to an edge node.
4. Perform related operations, such as monitoring the device status.

### 3.2.2 Device Templates

In an edge computing scenario, there are a large number of end devices. You can define a template for end devices of the same type and use it to register the devices on IEF. For example, you can create a device template named **Camera**. After you use this template to register end devices, these devices will use all properties defined in the template. In this way, you do not need to keep repeating the same configuration.

#### Creating a Device Template

**Step 1** Log in to the IEF console.

**Step 2** In the navigation pane, choose **Managed Resources > End Devices**. Then, click **Create Device Template** in the upper right corner.

**Step 3** Enter a template name, select an access protocol, and set template description, device properties, twin properties, and device tags.

**Figure 3-18** Creating a device template

- **Name:** name of a device template.
- **Access Protocol:** IEF supports **MQTT**.
- **Description:** description of the end device.
- **Device Property:**  
Properties are defined as key-value pairs. Enter a property name and value, and select a type.  
Metadata that does not change, such as serial numbers, asset identifiers, and MAC addresses, is defined in templates as device properties.
- **Twin Property:**  
Dynamic device data, such as dedicated real-time device data for a specific scenario, is set as twin properties. For example, the on and off states of a light are real-time data.
  - **MQTT:** Twin properties are defined as key-value pairs. Enter a property name and value, and select a type.

**NOTICE**

IEF does not provide any encryption or decryption tools, and does not sense your device attribute values. If the device attribute values are encrypted, you need to decrypt them.

- **Tags:** Tags are used to classify end devices. You can quickly search for desired devices by tag. Tags also facilitate end device management by category.

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

----End

## 3.2.3 End Devices

End devices can be connected to IEF through edge nodes by using the MQTT. After end devices are connected to IEF, you can manage them on IEF in a unified manner.

### Registering an End Device

**Step 1** Log in to the IEF console.

**Step 2** In the navigation pane, choose **Managed Resources > End Devices**. Then, click **Register End Device** in the upper right corner.

**Step 3** Set device parameters.

- **Name:** name of an end device.
- **ID:** You can customize a device ID or enable IEF to automatically generate an ID. If you want to customize a device ID, select **Custom device ID** and enter one.
- **Access Protocol:** IEF supports **MQTT**.
- **Description:** description of the end device.
- **Device Configuration:** Select an existing device template to add properties for the device automatically. The template defines device properties, twin properties, and device tags. You can also manually add device properties and tags without using a template. The meanings of the properties and tags manually added are the same as those defined in the template. For details, see [Device Templates](#).

**Step 4** Click **Register**.

----End

### Follow-up Procedure

You can bind end devices to edge nodes. For details, see [Binding an End Device to an Edge Node](#).

## 3.2.4 Binding an End Device to an Edge Node

An edge node can be bound to multiple end devices, but each end device can be bound to only one edge node. By binding an end device to an edge node, you can deploy applications on the edge node so that you can manage the device and monitor its status on IEF.

### Binding an Edge Node

**Step 1** Log in to the IEF console.

- Step 2** In the navigation pane, choose **Managed Resources > End Devices**.
- Step 3** Click **Bind Node** in the row where the end device is located.
- Step 4** Enter the relationship between the device and the node, select the edge node to be bound, and click **OK**.

----End

## Binding an End Device on the Edge Node Details Page

You can also bind an end device on the details page of an edge node.

- Step 1** Log in to the IEF console.
- Step 2** In the navigation pane, choose **Managed Resources > Edge Nodes**. Then, click an edge node name.
- Step 3** On the edge node details page, click the **End Devices** tab and then click **Bind**.
- Step 4** In the dialog box that is displayed, select the device to be bound, enter the relationship between the device and the node, and click **OK**.

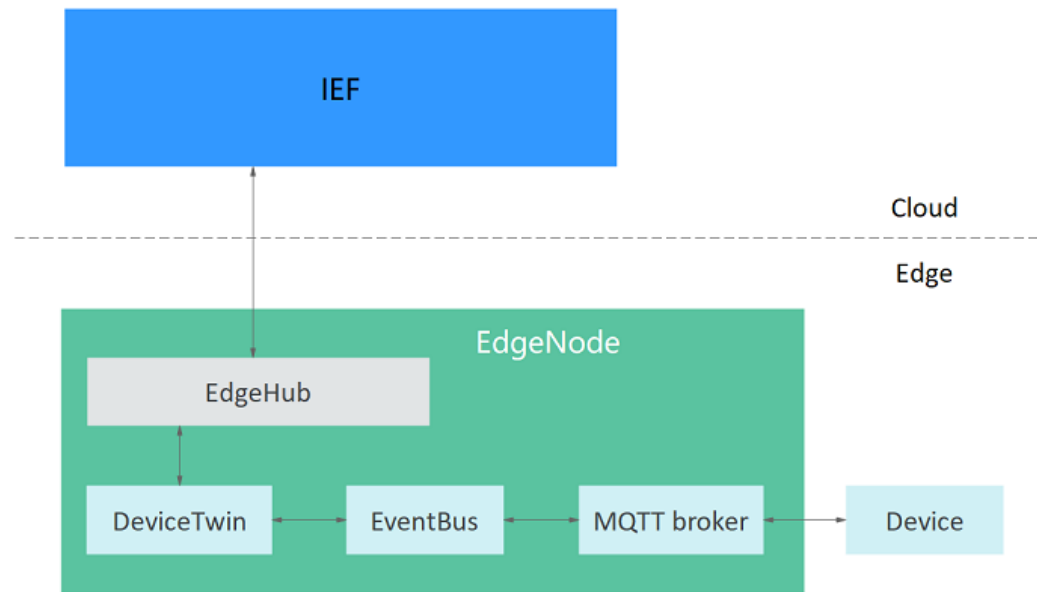
----End

## 3.2.5 Device Twin Working Principles

After an edge node is managed, Edge Agent is installed on the edge node. Edge Agent provides the following components for end device management:

- **EdgeHub**: a WebSocket client, which provides functions such as synchronizing cloud resource updates and reporting edge node and end device data to the cloud.
- **DeviceTwin**: stores end device status and synchronizes the end device status to the cloud.
- **EventBus**: a client that interacts with the MQTT server and provides the functions of subscribing to and publishing messages for other components.
- **MQTT broker**: an MQTT server.

**Figure 3-19** End device management



DeviceTwin plays an important role during the communication among end devices, edge nodes, and IEF. It maintains the dynamic device data, including the dedicated real-time data of devices in a specific scenario, such as the on/off status of lights.

DeviceTwin has the same features as physical devices and facilitates better communication between end devices and applications. The commands sent by an application reach DeviceTwin. Then, DeviceTwin updates the device status according to **Expected State** set by the application. In addition, the end device reports its **Actual State** in real time. DeviceTwin records both **Actual State** and **Expected State** of the end device. In this way, the status of an offline device can be synchronized when it comes back online.

The following is an example of a device twin:

```
{
  "device": {
    "id": "989e4fc8-9f24-44d7-9f9c-a0bd3bfb1949",
    "description": "my home light",
    "name": "light",
    "state": "online",
    "twin": {
      "powerstatus": {
        "expected": {
          "value": "ON"
        },
        "actual": {
          "value": "OFF"
        },
        "metadata": {
          "type": "string"
        }
      }
    }
  }
}
```

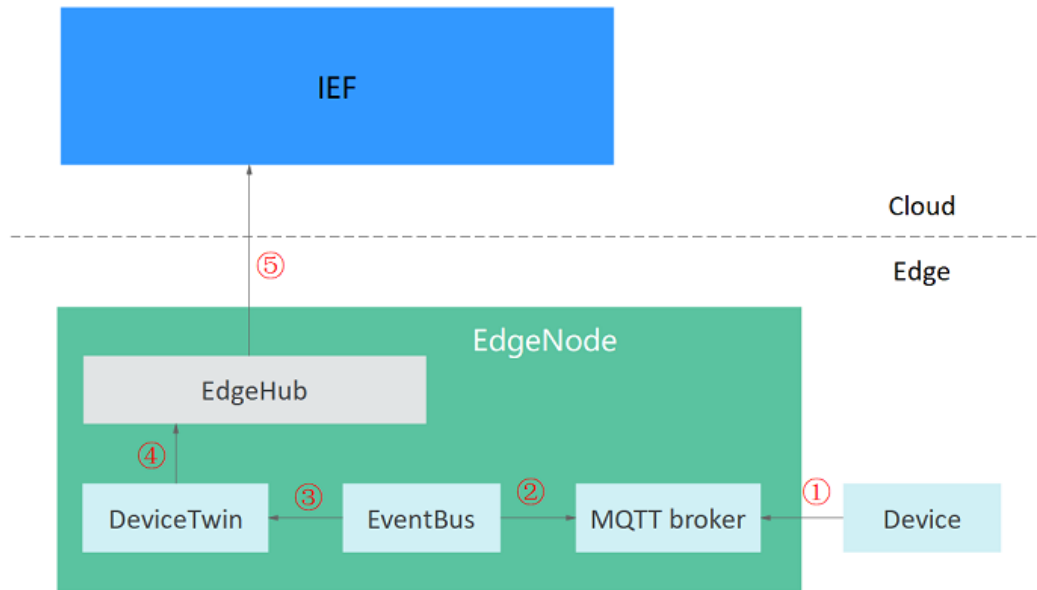
In this example, a twin property **powerstatus** is defined for the device named **light**, in which the expected value is **ON** and the actual value is **OFF**.

The following describes how this end device communicates with the edge node and IEF.

## Reporting the Actual Device Status to the Cloud

**Figure 3-20** shows the process for an end device to report its actual status to the cloud.

**Figure 3-20** Reporting the device status

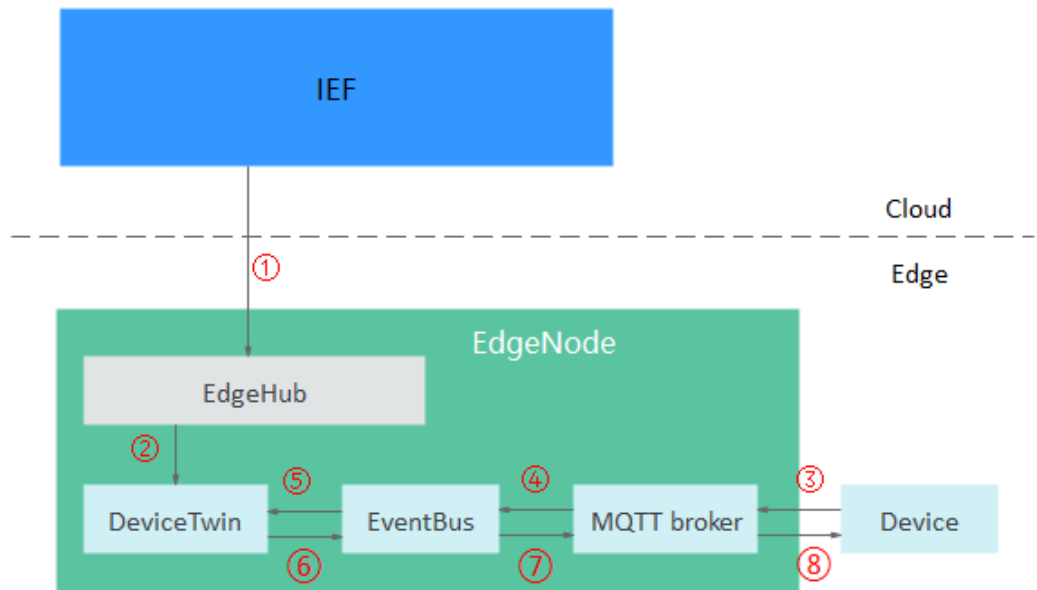


1. The end device reports its actual status to the MQTT broker in real time.
2. EventBus receives a subscription message from the MQTT broker. The message contains the actual status of the end device.
3. EventBus sends the actual device status to DeviceTwin. Then, DeviceTwin stores the actual device status on the edge node.
4. DeviceTwin synchronizes the actual status to EdgeHub (a WebSocket client).
5. EdgeHub sends a message to IEF.



## Modifying Twin Properties in the Cloud to Control Device Status

Figure 3-21 Modifying the device status



1. Modify the twin properties of the end device on IEF. Then, IEF sends the expected status of the end device to EdgeHub on the edge node.
2. EdgeHub sends a message containing the expected device status to DeviceTwin. Then, DeviceTwin stores the expected device status on the edge node.
3. The end device sends a message to the MQTT broker in real time to query the expected device status.
4. EventBus receives the message from the MQTT broker.
5. EventBus queries the expected device status based on the message.
6. DeviceTwin sends the expected device status to EventBus.
7. EventBus sends the expected device status to the MQTT broker.
8. The end device receives a subscription message from the MQTT broker and adjusts its status to the expected status.

### 3.2.6 Migrating Device Data to the Cloud

#### MQTT Broker

End devices can communicate with IEF using the MQTT protocol. You can also control end devices through pub/sub messaging.

An edge node has a **built-in MQTT broker**. The built-in MQTT broker uses port 8883 to communicate with end devices. End devices can communicate with the built-in MQTT broker only after they pass security authentication. For details, see [Performing Security Authentication Using Certificate](#).

Edge nodes also support **external MQTT brokers**. You can install an MQTT broker (for example, [Mosquitto](#), an open-source MQTT broker) on an edge node. Then devices can communicate with the edge node over port 1883.

 NOTE

If an external MQTT broker is used, ensure that the communication port of the external MQTT broker is available.

## MQTT Topics

The MQTT broker forwards messages between end devices and edge nodes to enable the end devices to communicate with the nodes and IEF. By default, the MQTT broker provides message topics described in [Table 3-4](#). Device status can be reported and controlled through pub/sub messaging.

After an application is compiled, you can deploy the application to an edge node through IEF. For details, see [Containerized Application Management](#).

**Table 3-4** Default topics provided by IEF

Name	Type	Topic	Description
<a href="#">Device Twin Update</a>	Subscribe	<code>\$hw/events/device/{device_id}/twin/update/document</code>	This topic is used to subscribe to device twin updates. It reflects the differences before and after a device twin update.
<a href="#">Device Twin Delta</a>	Subscribe	<code>\$hw/events/device/{device_id}/twin/update/delta</code>	This topic is used to subscribe to device twin delta events. When a device twin changes, the twin properties whose actual values are different from expected values are returned.
<a href="#">Device Member Update</a>	Subscribe	<code>\$hw/events/node/{node_id}/membership/updated</code>	This topic is used to subscribe to end device binding updates.
<a href="#">Device Property Update</a>	Subscribe	<code>\$hw/events/device/{device_id}/updated</code>	This topic is used to subscribe to end device property updates.
<a href="#">Device Member Acquisition</a>	Publish	<code>\$hw/events/node/{node_id}/membership/get</code>	This topic is used to subscribe to end device bindings.
<a href="#">Device Member Acquisition Result</a>	Subscribe	<code>\$hw/events/node/{node_id}/membership/get/result</code>	This topic is used to subscribe to the result of obtaining information about end device members.
<a href="#">Device Twin Acquisition</a>	Publish	<code>\$hw/events/device/{device_id}/twin/get</code>	This topic is used to publish the request for obtaining device twins.

Name	Type	Topic	Description
<b>Device Twin Acquisition Result</b>	Subscribe	\$hw/events/device/{device_id}/twin/get/result	This topic is used to subscribe to the result of obtaining device twins.
<b>Device Twin Modification</b>	Publish	\$hw/events/device/{device_id}/twin/update	This topic is used to publish device twin modifications.
<b>Device Twin Modification Result</b>	Subscribe	\$hw/events/device/{device_id}/twin/update/result	This topic is used to subscribe to the result of modifying device twins.
<b>Encryption Data Request</b>	Publish	\$hw/{project_id}/encryptdatas/{encryptdata_name}/properties/{properties_name}/decrypt	This topic is used to publish the request for obtaining encryption data.
<b>Encryption Data Acquisition</b>	Subscribe	\$hw/{project_id}/encryptdatas/{encryptdata_name}/properties/{properties_name}/plaintext	This topic is used to subscribe to encryption data.
<b>Alarm Reporting</b>	Publish	\$hw/alarm/{appname}/add	This topic is used to report alarms to AOM.
<b>Alarm Clearance</b>	Publish	\$hw/alarm/{appname}/clear	This topic is used to clear alarms in AOM.
<b>Custom Topics</b>	Publish	{project_id}/nodes/{node_id}/user/{custom_topic}	Topics customized based on actual requirements. You can publish end device data to a custom topic in the MQTT broker of an edge node. IEF then forwards the device data to a DIS stream or an API Gateway address. Then, you can extract the data for processing and analysis.

The following describes how to obtain end device data at the edge, receive control messages from the cloud, and report end device data to the cloud. For details

about the sample code for MQTT to send and receive messages, see [Go-Language Code Sample](#) and [Java-Language Code Sample](#).

## Obtaining End Devices Associated with an Edge Node

**Step 1** Send a request to the topic [Device Member Acquisition](#).

**Topic:** \$hw/events/node/{node\_id}/membership/get

**Payload:** {"event\_id": "Custom ID"}

For example:

```
$hw/events/node/{node_id}/membership/get  
{"event_id":""}
```

**Step 2** Subscribe to the topic [Device Member Acquisition Result](#).

**Topic:** \$hw/events/node/{node\_id}/membership/get/result

The following is an example of the returned result:

```
{  
  "event_id": "",  
  "timestamp": 1554986455386,  
  "devices": [  
    {  
      "id": "2144773f-13f1-43f5-af07-51991d4fd064",  
      "name": "equipmentA",  
      "state": "unknown",  
      "attributes": {  
        "name": {  
          "value": "a",  
          "optional": true,  
          "metadata": {  
            "type": "string"  
          }  
        }  
      }  
    }  
  ]  
}
```

----End

## Obtaining Device Twins

**Step 1** Send a request to the topic used to [Device Twin Acquisition](#).

**Topic:** \$hw/events/device/{device\_id}/twin/get

**Payload:** {"event\_id": "Custom ID"}

**Step 2** Subscribe to the topic [Device Twin Acquisition Result](#).

**Topic:** \$hw/events/device/{device\_id}/twin/get/result

The following is an example of the returned result:

```
{  
  "event_id": "",  
  "timestamp": 1554988425592,  
  "twin": {  
    "humidity": {  
      "expected": {  
        "value": "0",
```

```

        "metadata": {
          "timestamp": 1554988419529
        }
      },
      "optional": true,
      "metadata": {
        "type": "int"
      }
    },
    "temperature": {
      "expected": {
        "value": "0",
        "metadata": {
          "timestamp": 1554988419529
        }
      }
    },
    "optional": true,
    "metadata": {
      "type": "int"
    }
  }
}

```

----End

## Listening to Device Twin Events

After [Obtaining End Devices Associated with an Edge Node](#) and [Obtaining Device Twins](#) are performed, the ID of the end device bound to the node is obtained. Then, the events of the end device can be listened to.

Update the device twin by setting the expected value in the cloud to control the end device.

For example, a device has two twin properties: humidity and temperature.

**Figure 3-22** Twin properties

Name	Type	Expected Value	Created
humidity	int	9	Jul 13, 2020 09:40:38 GMT+08:00
temperature	int	0	Jul 13, 2020 09:40:58 GMT+08:00

Change the value of twin property **humidity** on the **Device Twins** tab page from **9** to **10**. After the twin property is changed, the device receives the following two events:

- **Device twin change event:** includes the twin information before and after the change.
- **Device twin delta event:** includes the device twin details and the delta information about the inconsistency between the expected and actual values of twin properties.

The device can receive the two events only after it has subscribed to the two topics.

**Step 1** Subscribe to the topic **Device Twin Update**.**Topic:** \$hw/events/device/{device\_id}/twin/update/document

The update message received at the edge is as follows:

```
{
  "event_id": "0f921313-4074-46a2-96f6-aac610721059",
  "timestamp": 1555313685831,
  "twin": {
    "humidity": {
      "last": {
        "expected": {
          "value": "9",
          "metadata": {
            "timestamp": 1555313665978
          }
        },
        "optional": true,
        "metadata": {
          "type": "int"
        }
      },
      "current": {
        "expected": {
          "value": "10",
          "metadata": {
            "timestamp": 1555313685831
          }
        },
        "optional": true,
        "metadata": {
          "type": "int"
        }
      }
    },
    "temperature": {
      "last": {
        "expected": {
          "value": "0",
          "metadata": {
            "timestamp": 1555313665978
          }
        },
        "actual": {
          "value": "2",
          "metadata": {
            "timestamp": 1555299457284
          }
        },
        "optional": true,
        "metadata": {
          "type": "int"
        }
      },
      "current": {
        "expected": {
          "value": "0",
          "metadata": {
            "timestamp": 1555313685831
          }
        },
        "actual": {
          "value": "2",
          "metadata": {
            "timestamp": 1555299457284
          }
        },
        "optional": true,
        "metadata": {

```

```

        "type": "int"
      }
    }
  }
}

```

**Step 2** Subscribe to the topic **Device Twin Delta**.

**Topic:** \$hw/events/device/{device\_id}/twin/update/delta

The update message received at the edge is as follows:

```

{
  "event_id": "60fb5baf-d4ad-47b0-a21e-8b57b52d0978",
  "timestamp": 1555313685837,
  "twin": {
    "humidity": {
      "expected": {
        "value": "10",
        "metadata": {
          "timestamp": 1555313685831
        }
      },
      "optional": true,
      "metadata": {
        "type": "int"
      }
    },
    "temperature": {
      "expected": {
        "value": "0",
        "metadata": {
          "timestamp": 1555313685831
        }
      },
      "actual": {
        "value": "2",
        "metadata": {
          "timestamp": 1555299457284
        }
      },
      "optional": true,
      "metadata": {
        "type": "int"
      }
    }
  },
  "delta": {
    "humidity": "10",
    "temperature": "0"
  }
}

```

----End

## Reporting Actual Values of Device Properties

**Step 1** Publish a twin update event for an end device.

**Topic:** \$hw/events/device/{device\_id}/twin/update

**Payload:** {"event\_id":"","timestamp":0,"twin":{"Property":{"actual":{"value":"Actual value"}}}}

For example:

```

{
  "event_id": "",

```

```

"timestamp": 0,
"twin": {
  "temperature": {
    "actual": {
      "value": "2"
    }
  }
}
}

```

After the event is published, you can find that the actual value of the device twin has changed in the cloud, as shown in [Figure 3-23](#).

**Figure 3-23** Updated twin properties

Name	Type	Expected Value	Created	Actual Value
humidity	int	9	Jul 13, 2020 09:40:38 GMT+08:00	
temperature	int	0	Jul 13, 2020 09:49:22 GMT+08:00	2

**Step 2** Subscribe to the topic **Device Twin Modification Result** at the edge. The edge node then can receive the result of the device twin update event.

**Topic:** \$hw/events/device/{device\_id}/twin/update/result

The update result is as follows:

```

{
  "event_id": "",
  "timestamp": 1554992093859,
  "twin": {
    "temperature": {
      "actual": {
        "value": "2",
        "metadata": {
          "timestamp": 1554992093859
        }
      }
    },
    "optional": true,
    "metadata": {
      "type": "int"
    }
  }
}
}

```

----End

## 3.2.7 Performing Security Authentication Using Certificate

### Scenario

By default, the built-in MQTT broker enables the port for Transport Layer Security (TLS) authentication. A client can access the MQTT broker only when it has a certificate.

End devices and applications can use the certificates added on the node details page for security authentication.



## Constraints

- Certificates are bound to edge nodes. The certificates applied for on an edge node can be used only to access the MQTT broker of this edge node. If these certificates are used to access the MQTT broker of another edge node, the authentication will fail.
- A maximum of 10 certificates can be applied for an edge node.
- The validity period of a certificate is 5 years.
- There are constraints on using MQTT.

**Table 3-5** MQTT constraints

Description	Restriction
Supported MQTT version	3.1.1
Differences from the standard MQTT protocol	<ul style="list-style-type: none"><li>• Quality of Service (QoS) 0 is supported.</li><li>• Topic customization is supported.</li><li>• QoS 1 and QoS 2 are not supported.</li><li>• <b>will</b> and <b>retain</b> messages are not supported.</li></ul>
MQTTS security levels	TCP channel basic + TLS protocol (TLS v1.2)

## Applying for a Certificate

### NOTE

The validity period of a certificate is 5 years.

**Step 1** Log in to the IEF console.

**Step 2** In the navigation pane, choose **Managed Resources > Edge Nodes**.

**Step 3** Click an edge node name. The edge node details page is displayed.

**Step 4** Click the **Certificates** tab, and click **Add Certificate**.

**Step 5** Enter a certificate name and click **OK**.

After the certificate is added, the system will automatically download the certificate file. Keep the certificate file secure.

**Figure 3-24** Adding a certificate

**Add Certificate** ×

**i** After the certificate is added successfully, the system automatically downloads the certificate file.

\* Name

Description  0/255

**OK**

----End

## Using a Certificate

A certificate is used for authentication when an end device communicates with the MQTT broker.

[Go-Language Code Sample](#) and [Java-Language Code Sample](#) illustrate how to use certificates for authentication.

### NOTE

- The client does not need to verify the server certificate. In other words, one-way authentication is required.
- Port 8883 of the built-in MQTT broker is enabled by default.
- In the Go-language code sample, the MQTT client references [github.com/eclipse/paho.mqtt.golang](https://github.com/eclipse/paho.mqtt.golang) (an open-source library).
- The MQTT client is required to process disconnection events and reestablish connections to improve the connection reliability.

## Go-Language Code Sample

```
package main

import (
    "crypto/tls"
    "crypto/x509"
    "fmt"
    "math/rand"
    "sync"
    "time"

    MQTT "github.com/eclipse/paho.mqtt.golang"
)
```

```
func main() {
    subClient := InitMqttClient(onSubConnectionLost)
    pubClient := InitMqttClient(onPubConnectionLost)

    wait := sync.WaitGroup{}
    wait.Add(1)

    go func() {
        for {
            time.Sleep(1*time.Second)
            pubClient.Publish("topic", 0, false, "hello world")
        }
    }()

    subClient.Subscribe("topic", 0, onReceived)

    wait.Wait()
}

func InitMqttClient(onConnectionLost MQTT.ConnectionLostHandler) MQTT.Client {
    pool := x509.NewCertPool()
    cert, err := tls.LoadX509KeyPair("/tmp/example_cert.crt", "/tmp/example_cert.key")
    if err != nil {
        panic(err)
    }

    tlsConfig := &tls.Config{
        RootCAs: pool,
        Certificates: []tls.Certificate{cert},
        // One-way authentication, that is, the client does not verify the server certificate.
        InsecureSkipVerify: true,
    }
    // Use the TLS or SSL protocol to connect to port 8883.
    opts := MQTT.NewClientOptions().AddBroker("tls://127.0.0.1:8883").SetClientID(fmt.Sprintf("%f", rand.Float64()))
    opts.SetTLSConfig(tlsConfig)
    opts.OnConnect = onConnect
    opts.AutoReconnect = false
    // Callback function. It is triggered immediately after the client is disconnected from the server.
    opts.OnConnectionLost = onConnectionLost
    client := MQTT.NewClient(opts)
    loopConnect(client)
    return client
}

func onReceived(client MQTT.Client, message MQTT.Message) {
    fmt.Printf("Receive topic: %s, payload: %s \n", message.Topic(), string(message.Payload()))
}

// The reconnection mechanism is triggered after the subscribe client is disconnected from the server.
func onSubConnectionLost(client MQTT.Client, err error) {
    fmt.Println("on sub connect lost, try to reconnect")
    loopConnect(client)
    client.Subscribe("topic", 0, onReceived)
}

// The reconnection mechanism is triggered after the publish client is disconnected from the server.
func onPubConnectionLost(client MQTT.Client, err error) {
    fmt.Println("on pub connect lost, try to reconnect")
    loopConnect(client)
}

func onConnect(client MQTT.Client) {
    fmt.Println("on connect")
}

func loopConnect(client MQTT.Client) {
    for {
```

```
token := client.Connect()
if rs, err := CheckClientToken(token); !rs {
    fmt.Printf("connect error: %s\n", err.Error())
} else {
    break
}
time.Sleep(1 * time.Second)
}
}

func CheckClientToken(token MQTT.Token) (bool, error) {
    if token.Wait() && token.Error() != nil {
        return false, token.Error()
    }
    return true, nil
}
```

## Java-Language Code Sample

The format of an MqttClientDemo.java file is as follows:

```
/*
Description: A java demo of MQTT message sending and receiving. You need to create an edge node and
download the client certificate.
*/

package com.example.demo;

import javax.net.ssl.SSLSocketFactory;

import org.eclipse.paho.client.mqttv3.IMqttDeliveryToken;
import org.eclipse.paho.client.mqttv3.MqttCallback;
import org.eclipse.paho.client.mqttv3.MqttClient;
import org.eclipse.paho.client.mqttv3.MqttConnectOptions;
import org.eclipse.paho.client.mqttv3.MqttException;
import org.eclipse.paho.client.mqttv3.MqttMessage;
import org.eclipse.paho.client.mqttv3.persist.MemoryPersistence;

/*
* MQTT demo shows how the client connects to the MQTT broker of the edge node to send and receive
messages and how SSL security authentication is performed on the connection. The demo illustrates the
following flows:
* 1. The MQTT subscribe client receives MQTT messages.
* 2. The MQTT publish client sends MQTT messages.
*/

public class MqttClientDemo {
    private static int QOS_TYPE = 2;
    //MQTT server address
    private static final String MQTT_HOST = "ssl://x.x.x.x:8883";
    //MQTT publish client ID
    private static final String MQTT_PUB_CLIENT_ID = "pub_client_1";
    //MQTT subscribe client ID
    private static final String MQTT_SUB_CLIENT_ID = "sub_client_1";
    //MQTT channel subscription topic
    private static final String TOPIC = "/hello";
    //Paths of the SSL certificate used for MQTT client connection
    public static final String CLIENT_CERT_FILE_PATH = "example_cert.crt";
    public static final String CLIENT_KEY_FILE_PATH = "example_cert.key";
    //MQTT client connection timeout interval (s)
    public static final int TIME_OUT_INTERVAL = 10;
    //Interval at which the MQTT client sends a heartbeat message, in seconds
    public static final int HEART_TIME_INTERVAL = 20;
    //Interval at which the MQTT client retries upon disconnection, in milliseconds
    public static final int RECONNECT_INTERVAL = 10000;
    //Interval at which the MQTT client sends a message, in seconds
    public static final int PUBLISH_MSG_INTERVAL = 3000;

    //MQTT client
    private MqttClient mqttClient;
```

```
//MQTT client ID.
private String clientId;
//MQTT client connection options
private MqttConnectOptions connOpts;
//Initialized MQTT client has not subscribed to any topic.
private boolean isSubscribe = false;

public MqttClientDemo(String id) throws MqttException {
    setClientId(id);
    initMqttClient();
    initCallback();
    initConnectOptions();
    connectMqtt();
}

/*****
 * Sending messages
 * @param message Message to be sent
 * @throws MqttException
 *****/
public void publishMessage(String message) throws MqttException {
    MqttMessage mqttMessage = new MqttMessage(message.getBytes());
    mqttMessage.setQos(QOS_TYPE);
    mqttMessage.setRetained(false);
    mqttClient.publish(TOPIC, mqttMessage);
    System.out.println(String.format("MQTT Client[%s] publish message[%s]", clientId, message));
}

/*****
 * Subscribing to topics
 * @throws MqttException
 *****/
public void subscribeTopic() throws MqttException {
    int[] Qos = {QOS_TYPE};
    String[] topics = {TOPIC};
    mqttClient.subscribe(topics, Qos);
    isSubscribe = true;
}

/*****
 * Starting the thread to periodically send MQTT messages
 * @throws MqttException
 *****/
public void startPublishMessage() {
    new Thread() {
        @Override
        public void run() {
            while (true) {
                try {
                    Thread.sleep(PUBLISH_MSG_INTERVAL);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                try {
                    publishMessage("hello world!");
                } catch (MqttException e) {
                    System.out.println(String.format("MQTT client[%s] publish message error,errorMsg[%s]",
clientId, e.getMessage()));
                }
            }
        }
    }.start();
}

/*****
 * Initializing the MQTT client
 * @throws MqttException Abnormal connection
 *****/
private void initMqttClient() throws MqttException {
```

```
MemoryPersistence persistence = new MemoryPersistence();
mqttClient = new MqttClient(MQTT_HOST, clientId, persistence);
}

/*****
 * Initializing connection options
 * @throws MqttException Abnormal connection
 *****/
private void initConnectOptions() {
    connOpts = new MqttConnectOptions();
    // Specify whether to clear the session. If the value is false, the server retains the client connection
    records. If the value is true, the client connects to the server as a new client.
    connOpts.setCleanSession(true);
    connOpts.setHttpsHostnameVerificationEnabled(false);
    // Set the timeout interval, in seconds.
    connOpts.setConnectionTimeout(TIME_OUT_INTERVAL);
    // Set the interval at which a session heartbeat message is sent, in seconds. The server sends a
    message to the client every 1.5 x 20 seconds to check whether the client is online. However, this method
    does not provide the reconnection mechanism.
    connOpts.setKeepAliveInterval(HEART_TIME_INTERVAL);
    SSLSocketFactory factory = null;
    try {
        factory = SslUtil.getSocketFactory(CLIENT_CERT_FILE_PATH, CLIENT_KEY_FILE_PATH);
    } catch (Exception e) {
        e.printStackTrace();
    }
    // TLS connection configuration
    connOpts.setSocketFactory(factory);
}

/*****
 * Initiate an MQTT connect request.
 * @throws MqttException Abnormal connection
 *****/
private void connectMqtt() throws MqttException {
    mqttClient.connect(connOpts);
    System.out.println(String.format("MQTT client[%s] is connected,the connOptions: \n%s", clientId,
connOpts.toString()));
}

/*****
 * Set the callback API.
 * @throws MqttException Abnormal connection
 *****/
private void initCallback() {
    mqttClient.setCallback(new MqttMessageCallback());
}

private void setClientId(String id) {
    clientId = id;
}

/*****
 * MQTT client reconnection function. This function is called to check whether a topic has been
 subscribed to. If yes, the topic will be re-subscribed to.
 * @throws MqttException
 *****/
private void reconnectMqtt() throws MqttException {
    connectMqtt();
    if (isSubscribe) {
        subscribeTopic();
    }
}

/*****
 * After the MQTT client subscribes to topics, the MQTT client receives messages through the callback
 API if the MQTT channel has data.
 * @version V1.0
 *****/
```

```
private class MqttMessageCallback implements MqttCallback {

    @Override
    public void connectionLost(Throwable cause) {
        System.out.println(String.format("MQTT Client[%s] connect lost,Retry in 10 seconds,info[%s]",
clientId, cause.getMessage()));
        while (!mqttClient.isConnected()) {
            try {
                Thread.sleep(RECONNECT_INTERVAL);
                System.out.println(String.format("MQTT Client[%s] reconnect ....", clientId));
                rconnectMqtt();
            } catch (Exception e) {
                continue;
            }
        }
    }

    @Override
    public void messageArrived(String topic, MqttMessage mqttMessage) {
        String message = new String(mqttMessage.getPayload());
        System.out.println(String.format("MQTT Client[%s] receive message[%s] from topic[%s]", clientId,
message, topic));
    }

    @Override
    public void deliveryComplete(IMqttDeliveryToken iMqttDeliveryToken) {

    }
}

public static void main(String[] args) throws MqttException {
    try {
        //Subscribe to the MQTT channel.
        MqttClientDemo mqttsubClientDemo = new
MqttClientDemo(MqttClientDemo.MQTT_SUB_CLIENT_ID);
        mqttsubClientDemo.subscribeTopic();
        //Send hello world to the MQTT channel.
        MqttClientDemo mqttpubClientDemo = new
MqttClientDemo(MqttClientDemo.MQTT_PUB_CLIENT_ID);
        mqttpubClientDemo.startPublishMessage();
    } catch (MqttException e) {
        System.out.println(String.format("program start error,errorMessage[%s]", e.getMessage()));
    }
}
}
```

The format of an SSLUtil.java file is as follows:

```
/*
Description: SSL utility class. Load the client SSL certificate configuration and ignore server certificate
verification.
*/

package com.example.demo;

import java.io.ByteArrayInputStream;
import java.io.InputStreamReader;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.security.KeyPair;
import java.security.KeyStore;
import java.security.Security;
import java.security.cert.Certificate;
import java.security.cert.CertificateException;
import java.security.cert.X509Certificate;

import javax.net.ssl.KeyManagerFactory;
import javax.net.ssl.SSLContext;
```

```
import javax.net.ssl.SSLSocketFactory;
import javax.net.ssl.TrustManager;
import javax.net.ssl.X509TrustManager;

import org.bouncycastle.jce.provider.BouncyCastleProvider;
import org.bouncycastle.openssl.PEMReader;
import org.bouncycastle.openssl.PasswordFinder;

public class SslUtil {

    /*****
     * Verify and obtain the SSLSocketFactory.
     *****/
    public static SSLSocketFactory getSocketFactory(final String crtFile, final String keyFile) throws Exception
    {
        Security.addProvider(new BouncyCastleProvider());

        // 1. Load the client certificate.
        PEMReader reader_client =
            new PEMReader(new InputStreamReader(new
                ByteArrayInputStream(Files.readAllBytes(Paths.get(crtFile))));
            X509Certificate cert = (X509Certificate) reader_client.readObject();
            reader_client.close();

        // 2. Load the client key.
        reader_client = new PEMReader(
            new InputStreamReader(new ByteArrayInputStream(Files.readAllBytes(Paths.get(keyFile))),
            new PasswordFinder() {
                @Override
                public char[] getPassword() {
                    return null;
                }
            }
        );

        // 3. Send the client key and certificate to the server for identity authentication.
        KeyStore ks = KeyStore.getInstance(KeyStore.getDefaultType());
        ks.load(null, null);
        ks.setCertificateEntry("certificate", cert);
        ks.setKeyEntry("private-key", ((KeyPair) reader_client.readObject()).getPrivate(), "".toCharArray(), new
        Certificate[] {cert});
        KeyManagerFactory kmf =
        KeyManagerFactory.getInstance(KeyManagerFactory.getDefaultAlgorithm());
        kmf.init(ks, "".toCharArray());

        // 4. Create a socket factory.
        SSLContext context = SSLContext.getInstance("TLSv1.2");
        TrustManager[] tms = new TrustManager[1];
        TrustManager miTM = new TrustAllManager();
        tms[0] = miTM;
        context.init(kmf.getKeyManagers(), tms, null);

        reader_client.close();

        return context.getSocketFactory();
    }

    /*****
     * Ignore server certificate verification.
     *****/
    static class TrustAllManager implements TrustManager, X509TrustManager {
        @Override
        public X509Certificate[] getAcceptedIssuers() {
            return null;
        }

        @Override
```



```
public void checkServerTrusted(X509Certificate[] certs, String authType)
    throws CertificateException {
}

public boolean isServerTrusted(X509Certificate[] certs) {
    return true;
}

public boolean isClientTrusted(X509Certificate[] certs) {
    return true;
}

@Override
public void checkClientTrusted(X509Certificate[] certs, String authType)
    throws CertificateException {
}
}
```

The format of a pom.xml file is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.example</groupId>
    <artifactId>mqtt.example</artifactId>
    <version>1.0-SNAPSHOT</version>
    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <configuration>
                    <source>7</source>
                    <target>7</target>
                </configuration>
            </plugin>
        </plugins>
    </build>
    <dependencies>
        <!-- https://mvnrepository.com/artifact/org.eclipse.paho/org.eclipse.paho.client.mqttv3 -->
        <dependency>
            <groupId>org.eclipse.paho</groupId>
            <artifactId>org.eclipse.paho.client.mqttv3</artifactId>
            <version>1.2.1</version>
        </dependency>
        <!-- https://mvnrepository.com/artifact/org.bouncycastle/bcprov-jdk16 -->
        <dependency>
            <groupId>org.bouncycastle</groupId>
            <artifactId>bcprov-jdk16</artifactId>
            <version>1.45</version>
        </dependency>
    </dependencies>
</project>
```

## 3.2.8 MQTT Topics

### 3.2.8.1 Device Twin Update

This topic is used to subscribe to device twin updates. It reflects the differences before and after a device twin update.

## Topic

`$hw/events/device/{device_id}/twin/update/document`

Parameter	Type	Description
device_id	String	End device ID.

## Usage

Use an MQTT client to subscribe to this topic.

## Parameter Description

Parameter	Type	Description
event_id	String	Event ID.
timestamp	Int64	Timestamp when the event occurred.
twin	Object	A set of updated device twins. Each twin is in the key-value pair format. The value contains <b>last</b> (twin information before the update) and <b>last</b> (twin information after the update). The twin information contains the option flag, the twin metadata contains the value type, the expected status contains the expected value and update time, and the actual status contains the actual value and update time.

## Example

When an end device is bound to an edge node, the following message is received:

```
$hw/events/device/{device_id}/twin/update/document
{
  "event_id": "",
  "timestamp": 1557314742122,
  "twin": {
    "state": {
      "last": null,
      "current": {
        "expected": {
          "value": "running",
          "metadata": {
            "timestamp": 1557314742122
          }
        }
      },
      "optional": true,
      "metadata": {
        "type": "string"
      }
    }
  }
}
```

### 3.2.8.2 Device Twin Delta

This topic is used to subscribe to device twin delta events. When a device twin changes, the twin properties whose actual values are different from expected values are returned.

#### Topic

`$hw/events/device/{device_id}/twin/update/delta`

Parameter	Type	Description
device_id	String	End device ID.

#### Usage

Use an MQTT client to subscribe to this topic.

#### Parameter Description

Parameter	Type	Description
event_id	String	Event ID.
timestamp	Int64	Timestamp when the event occurred.
twin	Object	A set of updated device twins. Each twin is in the key-value pair format. The value contains the option flag, the twin metadata contains the value type, the expected status contains the expected value and update time, and the actual status contains the actual value and update time.
delta	Map	Names of twin properties whose actual values are different from expected values, and expected values of these properties.

#### Example

When an end device is bound to an edge node, the following message is received:

```
$hw/events/device/{device_id}/twin/update/delta
{
  "event_id":"b9625811-f34f-4252-bee9-98185e7e1ec7",
  "timestamp":1557314742131,
  "twin":{
    "state":{
      "expected":{
        "value":"running",
        "metadata":{
          "timestamp":1557314742122
        }
      }
    }
  }
}
```

```

    },
    "optional":true,
    "metadata":{
      "type":"string"
    }
  }
},
"delta":{
  "state":"running"
}
}

```

### 3.2.8.3 Device Member Update

This topic is used to subscribe to updates on bindings between end devices and edge nodes.

#### Topic

\$hw/events/node/{node\_id}/membership/updated

Parameter	Type	Description
node_id	String	Node ID.

#### Usage

Use an MQTT client to subscribe to this topic.

#### Parameter Description

Parameter	Type	Description
event_id	String	Event ID.
timestamp	Int64	Timestamp when the event occurred.
added_devices	Array	Information about end devices, including the device ID, name, properties, and twin properties.
removed_devices	Array	Information about end devices, including the device ID, name, properties, and twin properties.

#### Example

When an end device is bound to an edge node, the following message is received:

```

$hw/events/node/{node_id}/membership/updated
{
  "event_id":"04a975ab-fd51-49be-85f5-5967e994f640",
  "timestamp":1557314742136,
  "added_devices":[
    {
      "id":"ab39361a-6fc0-4c94-b919-72b1e08ca690",
      "name":"IEF-device",

```

```

"state":"unknown",
"attributes":{
  "address":{
    "value":"xxx",
    "optional":true,
    "metadata":{
      "type":"string"
    }
  }
},
"twin":{
  "state":{
    "expected":{
      "value":"running",
      "metadata":{
        "timestamp":1557314434570
      }
    },
    "optional":true,
    "metadata":{
      "type":"string"
    }
  }
},
"removed_devices":null
}

```

### 3.2.8.4 Device Property Update

This topic is used to subscribe to updates on end device properties.

#### Topic

\$hw/events/device/{device\_id}/updated

Parameter	Type	Description
device_id	String	End device ID.

#### Usage

Use an MQTT client to subscribe to this topic.

#### Parameter Description

Parameter	Type	Description
event_id	String	Event ID.
timestamp	Int64	Timestamp when the event occurred.
attributes	Object	A set of updated end device properties, which are in the key-value pair format. The key indicates the property name, the value contains the property value and the optional flag, and metadata contains the value type.

## Example

When an end device is bound to an edge node, the following message is received:

```
$hw/events/device/{device_id}/updated
{
  "event_id": "",
  "timestamp": 1557314742136,
  "attributes": {
    "address": {
      "value": "xxx",
      "optional": true,
      "metadata": {
        "type": "string"
      }
    }
  }
}
```

### 3.2.8.5 Device Member Acquisition

This topic is used to publish the request for obtaining end device member information.

## Topic

\$hw/events/node/{node\_id}/membership/get

Parameter	Type	Description
node_id	String	Node ID.

## Usage

Use an MQTT client to publish this topic. This topic must be used together with [Device Member Acquisition Result](#).

## Parameter Description

Parameter	Type	Description
event_id	String	Event ID, which is user-defined.

## Example

```
$hw/events/node/3fbb5b8d-32db-4271-a34f-a013e021b6ce/membership/get
{
  "event_id": "bc876bc-345d-4050-86a8-319a5b13cc10"
}
```

### 3.2.8.6 Device Member Acquisition Result

This topic is used to subscribe to the result of obtaining end device member information.

## Topic

`$hw/events/node/{node_id}/membership/get/result`

Parameter	Type	Description
node_id	String	Node ID.

## Usage

Use an MQTT client to subscribe to this topic. This topic must be used together with [Device Member Acquisition](#).

## Parameter Description

Parameter	Type	Description
event_id	String	Event ID.
timestamp	Int64	Timestamp when the event occurred.
devices	Array	Information about end devices, including the device ID, name, and properties. The properties are in the key-value pair format. The key indicates the property name, the value contains the property value and the optional flag, and metadata contains the value type.

## Example

```
$hw/events/node/3fbb5b8d-32db-4271-a34f-a013e021b6ce/membership/get/result
{
  "event_id":"bc876bc-345d-4050-86a8-319a5b13cc10",
  "timestamp":1557317193524,
  "devices":[
    {
      "id":"ab39361a-6fc0-4c94-b919-72b1e08ca690",
      "name":"IEF-device",
      "state":"unknown",
      "attributes":{
        "address":{
          "value":"longgang",
          "optional":true,
          "metadata":{
            "type":"string"
          }
        }
      }
    }
  ]
}
```

### 3.2.8.7 Device Twin Acquisition

This topic is used to publish the request for obtaining device twins.

## Topic

\$hw/events/device/{device\_id}/twin/get

Parameter	Type	Description
device_id	String	End device ID.

## Usage

Use an MQTT client to publish this topic. This topic must be used together with [Device Twin Acquisition Result](#).

## Parameter Description

Parameter	Type	Description
event_id	String	Event ID.

## Example

```
$hw/events/device/ab39361a-6fc0-4c94-b919-72b1e08ca690/twin/get
{
  "event_id": "123456"
}
```

### 3.2.8.8 Device Twin Acquisition Result

This topic is used to subscribe to the result of obtaining device twins.

## Topic

\$hw/events/device/{device\_id}/twin/get/result

Parameter	Type	Description
device_id	String	Device ID.

## Usage

Use an MQTT client to subscribe to this topic. This topic must be used together with [Device Twin Acquisition](#).

## Parameter Description

Parameter	Type	Description
event_id	String	Event ID.



Parameter	Type	Description
timestamp	Int64	Timestamp when the event occurred.
twin	Object	A set of end device twin information. Each twin is in the key-value pair format. The value contains the option flag, the twin metadata contains the value type, the expected status contains the expected value and update time, and the actual status contains the actual value and update time.

## Example

```
$hw/events/device/ab39361a-6fc0-4c94-b919-72b1e08ca690/twin/get/result
{
  "event_id":"123456",
  "timestamp":1557317510926,
  "twin":{
    "state":{
      "expected":{
        "value":"stop",
        "metadata":{
          "timestamp":1557316778931
        }
      },
      "optional":true,
      "metadata":{
        "type":"string"
      }
    }
  }
}
```

### 3.2.8.9 Device Twin Modification

This topic is used to publish device twin modifications.

## Topic

```
$hw/events/device/{device_id}/twin/update
```

Parameter	Type	Description
device_id	String	End device ID.

## Usage

Use an MQTT client to publish this topic. This topic must be used together with [Device Twin Modification Result](#).

## Parameter Description

Parameter	Type	Description
event_id	String	Event ID.
timestamp	Int64	Timestamp when the event occurred.
twin	Object	A set of device twin information to be modified. The twin property is in the key-value pair format. The key is the name of the twin property to be modified, and the value contains the expected value of the expected status to be modified or the actual value of the actual status to be modified.

## Example

```
$hw/events/device/ab39361a-6fc0-4c94-b919-72b1e08ca690/twin/update
{
  "event_id":"123457",
  "twin":{
    "state":{
      "actual":{
        "value":"stop"
      }
    }
  }
}
```

### 3.2.8.10 Device Twin Modification Result

This topic is used to subscribe to the result of modifying device twins.

## Topic

```
$hw/events/device/{device_id}/twin/update/result
```

Parameter	Type	Description
device_id	String	End device ID.

## Usage

Use an MQTT client to subscribe to this topic. This topic must be used together with [Device Twin Modification](#).

## Parameter Description

Parameter	Type	Description
event_id	String	Event ID.

Parameter	Type	Description
timestamp	Int64	Timestamp when the event occurred.
twin	Object	A set of modified device twins. Each twin is in the key-value pair format. The value contains the option flag, the twin metadata contains the value type, the expected status contains the expected value and update time, and the actual status contains the actual value and update time.

## Example

```
$hw/events/device/ab39361a-6fc0-4c94-b919-72b1e08ca690/twin/update/result
```

```
{
  "event_id": "123457",
  "timestamp": 1557317614026,
  "twin": {
    "state": {
      "actual": {
        "value": "stop",
        "metadata": {
          "timestamp": 1557317614026
        }
      }
    },
    "optional": true,
    "metadata": {
      "type": "string"
    }
  }
}
```

### 3.2.8.11 Encryption Data Request

This topic is used to publish the request for obtaining encryption data.

## Topic

```
$hw/{project_id}/encryptdatas/{encryptdata_name}/properties/{properties_name}/  
decrypt
```

Parameter	Type	Description
project_id	String	Project ID. For details on how to obtain a project ID, see <a href="#">Obtaining a Project ID</a> .
encryptdata_name	String	Name of an encryption data record.
properties_name	String	Key of an encryption item.

## Usage

Use an MQTT client to publish this topic. This topic must be used together with [Encryption Data Acquisition](#).

A certificate must be used for security authentication when encryption data is requested. For details about the authentication method, see [Performing Security Authentication Using Certificate](#).

## Parameter Description

None.

## Example

Publish an empty message.

### 3.2.8.12 Encryption Data Acquisition

This topic is used to subscribe to encryption data.

## Topic

`$hw/{project_id}/encryptdatas/{encryptdata_name}/properties/{properties_name}/plaintext`

Parameter	Type	Description
project_id	String	Project ID. For details on how to obtain a project ID, see <a href="#">Obtaining a Project ID</a> .
encryptdata_name	String	Name of an encryption data record.
properties_name	String	Key of an encryption item.

## Usage

Use an MQTT client to subscribe to this topic. This topic must be used together with [Encryption Data Request](#).

A certificate must be used for security authentication when encryption data is requested. For details about the authentication method, see [Performing Security Authentication Using Certificate](#).

## Parameter Description

Parameter	Type	Description
plain_text	String	Plaintext value of an encryption item.

Parameter	Type	Description
ret_message	String	Error message.

## Example

If the request is correct, the following message is received:

```
{
  "ret_code": 200,
  "plain_text": "xxxxxxxxxx"
}
```

If the request is incorrect, the following message is received:

```
{
  "ret_code": 400,
  "ret_message": "xxxxxxxxxx"
}
```

### 3.2.8.13 Alarm Reporting

This topic is used to report alarms to AOM.

## Topic

\$hw/alarm/{appname}/add

Parameter	Type	Description
appname	String	Application name, which is user-defined.

## Usage

Use an MQTT client to publish this topic.

## Parameter Description

Parameter	Type	Description
alarmName	String	Alarm name.
alarmId	String	Alarm ID, which must be unique. For details, see <a href="#">Generating an alarmId</a> .
detailedInformation	String	Alarm description.
url	String	URL for root cause analysis. If the URL is not available, leave this field blank.

Parameter	Type	Description
source	String	Alarm source. The value is a character string consisting of uppercase and lowercase letters.
cleared	Boolean	Whether the alarm is cleared. <ul style="list-style-type: none"> <li>● <b>true</b>: The alarm has been cleared.</li> <li>● <b>false</b>: The alarm needs to be cleared manually.</li> </ul>
policyID	String	ID of an alarm rule. For a threshold rule, set this parameter to <b>ruleId</b> . If no threshold rule is configured, leave this field blank.
objectInstance	String	Location information. If this field is not specified, the value of this parameter is the same as that of <b>alarmId</b> by default.
perceivedSeverity	Integer	Alarm severity. <ul style="list-style-type: none"> <li>● 1: Critical</li> <li>● 2: Major</li> <li>● 3: Minor</li> <li>● 4: Warning</li> </ul>
resourceId	Object	Alarm information. For details, see <a href="#">Table 3-6</a> .
resourceType	String	Type of the resource for which the alarm is generated. <ul style="list-style-type: none"> <li>● Application</li> <li>● DB</li> <li>● Host</li> </ul>
eventType	Integer	Alarm type. <ul style="list-style-type: none"> <li>● 21: dynamic threshold alarm</li> <li>● 22: batch threshold alarm</li> <li>● 23: threshold alarm</li> <li>● 24: system alarm</li> <li>● 25: probe added or deleted</li> <li>● 26: Agent installation alarm</li> <li>● 27: Quota threshold-crossing alarm</li> </ul>
probableCause	String	Possible causes.
proposedRepairActions	String	Handling suggestions.

**Table 3-6** resourceId

Parameter	Type	Description
namespace	String	Resource type. The options are as follows: <ul style="list-style-type: none"> <li>● <b>PAAS.CONTAINER</b>: container metrics</li> <li>● <b>PAAS.NODE</b>: node metrics</li> </ul>
dimension	Object	Dimension information, which is associated with the node application information reported by the monitoring module. For details, see <a href="#">Table 3-7</a> . Currently, alarms are about nodes and applications. Therefore, you only need to pay attention to these two dimensions. The application dimension covers service, instance, container, and process information. You can select one or more types of information to be reported. <ul style="list-style-type: none"> <li>● Service information: clusterId, nameSpace, and serviceID</li> <li>● Instance information: podID and podName</li> <li>● Container information: containerID and containerName</li> <li>● Process information: processID and processName</li> </ul>

**Table 3-7** Dimension

Parameter	Type	Description
clusterId	String	Project ID. For details on how to obtain a project ID, see <a href="#">Obtaining a Project ID</a> .
nameSpace	String	The default value is <b>default</b> .
nodeIP	String	Node IP address.

Parameter	Type	Description
serviceID	String	<p>Service ID.</p> <ul style="list-style-type: none"> <li>For applications, the value is the MD5 value of {projectId}_{hostid}_{appName}.</li> <li>For processes, the value is the MD5 value of {projectId}_{hostid}_{Process name}_{Process pid}.</li> <li><b>processID</b> can be calculated as follows: md5({projectId})_{hostid}_md5({Process name})_{process pid}. <b>md5(projectId)</b> indicates the MD5 value of <b>projectId</b>, and <b>md5({Process name})</b> indicates the MD5 value of the process name.</li> </ul> <p>The dimension settings must be consistent with those reported by the monitoring module. Otherwise, the corresponding resources cannot be associated.</p>
podID	String	Instance ID.
podName	String	Instance name.
containerID	String	Container ID.
containerName	String	Container name.
processID	String	Process ID.
processName	String	Process name.
Application	String	Application name.

## Generating an alarmId

### NOTE

You do not need to use this method to generate the alarmId. However, you need to ensure that the alarmId is unique.

Generate the MD5 value of **{projectId}\_{Service name for which the alarm is generated}\_{Dimension information}\_{Metric name}\_{Alarm type}\_{Rule information}**.

Where:

- **projectId**: Project ID. For details on how to obtain a project ID, see [Obtaining a Project ID](#).
- **Dimension information**:
  - Node information: {clusterId}\_{namespace}\_{ip}



- Container information: {clusterId}\_{namespace}\_{appName}\_{podName}\_{containerId}
- Application information: {clusterId}\_{namespace}\_{appName}
- **Alarm type:**
  - 21: dynamic threshold alarm
  - 22: batch threshold alarm
  - 23: threshold alarm
  - 24: system alarm
  - 25: probe added or deleted
  - 26: Agent installation alarm
- **Rule information:** Set this field to a rule name for threshold alarms and to **NA** for alarms generated by the service itself. For dynamic threshold alarms, set this field to **policyId**.

## Example

- Node alarm

```
{
  "alarmName": "test",
  "alarmId": "73ccbccc05de74f9d3dda42f6ecfe20",
  "detailedInformation": "test",
  "url": "",
  "source": "IEF",
  "cleared": false,
  "policyID": "",
  "perceivedSeverity": 4,
  "resourceId": {
    "namespace": "PAAS.NODE",
    "dimension": {
      "clusterId": "e277befa37a64ed1aa25b522e686bc28",
      "nameSpace": "default",
      "nodeIP": "192.168.0.164"
    }
  }
},
"neType": "Host",
"eventType": 23
}
```

- Application alarm:

```
{
  "alarmName": "Application restart",
  "alarmId": "b09076ff565c59d4da0db0c9223781",
  "detailedInformation": "Application restart test",
  "url": "",
  "source": "IEF",
  "cleared": false,
  "policyID": "",
  "perceivedSeverity": 3,
  "resourceId": {
    "namespace": "PAAS.CONTAINER",
    "dimension": {
      "containerName": "container-e991acd3-864c-4038-8a90-e042eebab496",
      "containerID": "70b385315c8ac507b3de7dfe1258932cea0b53a850b7d030ce7ed0a55c47877c",
      "podID": "0e9ce4fd-b732-11e9-8a30-fa163e9b3546",
      "podName": "hvkapp1-7898f5bd4b-2lj8z"
    }
  }
},
"neType": "Application",
"eventType": 23
}
```

### 3.2.8.14 Alarm Clearance

This topic is used to clear alarms in AOM. The message body is the same as that described in [Alarm Reporting](#). To be specific, the only difference between the requests for reporting and clearing alarms is the topic name.

#### Topic

\$hw/alarm/{appname}/clear

Parameter	Type	Description
appname	String	Application name, which is user-defined.

#### Usage

Use an MQTT client to publish this topic.

#### Parameter Description

See [Alarm Reporting](#).

### 3.2.8.15 Custom Topics

IEF allows you to customize topics based on actual requirements.

You can publish end device data to a custom topic in the MQTT broker of an edge node. IEF then forwards the device data to a DIS stream or an API Gateway address. Then, you can extract the data for processing and analysis.

To use a custom topic, create a message route in which a topic is defined on the IEF console. IEF will forward the data in the topic based on the message route. For details on how to create a message route, see [Edge-Cloud Message Overview](#).

#### Topic

{project\_id}/nodes/{node\_id}/user/{custom\_topic}

#### Usage

Use an MQTT client to publish this topic.

#### Parameter Description

IEF can transparently forward any content in a custom topic.

**{custom\_topic}** supports the wildcards # and +. Therefore, multiple messages that meet a wildcard rule can be forwarded in a unified manner.

# is a wildcard that matches any number (0 or greater) of levels in a topic. + is a wildcard that matches only one level in a topic.

IEF performs the minimum match on the topic messages that match the wildcard rule and then forwards the messages. For example, you can configure 123/+/567 as a wildcard rule for message topics 123/aaa/567 and 123/bbb/567.

## Example

After a route is created, you can publish messages (data) to the custom topic defined in the route. IEF then forwards the messages to the specified endpoint. IEF also records the number of forwarded messages, including successful and failed messages.

**Figure 3-25** Number of forwarded messages

Route Na...	Source Endpoint	Destination Endpoint	Status	Forwarded Messages
asaadaaaa	SystemREST Cloud	service-8090 Edge	Enabled	Total: 0
--	/ada	/asdaaa		Successful: 0 Failed: 0

## 3.3 Containerized Application Management

### 3.3.1 Containerized Applications

IEF allows you to deliver custom edge applications to edge nodes. This section describes how to create a custom edge application.

#### Constraints

- When the disk usage of an edge node exceeds 70%, the image reclamation mechanism is started to reclaim the disk space occupied by the container image. In this case, if you deploy a containerized application, the application container will take a long time to start. Therefore, properly plan the disk space of the edge node before deploying the containerized application.
- When you create a containerized application, the edge node needs to pull an image from SWR. If the image size is too large and the download bandwidth of the edge node is small, the container image fails to be pulled. In this case, a message will be displayed on the IEF console, indicating that the containerized application fails to be created. However, the image pull will not be interrupted. After the container image is pulled successfully, the containerized application will be automatically created. To prevent such a problem, you can pull the container image to the edge node and then create a containerized application.
- The architecture of the container image to be used must be the same as that of the edge node on which a containerized application is to be deployed. For example, if the edge node uses x86, the container image must also use x86.

#### Creating an Edge Application

**Step 1** Log in to the IEF console.

**Step 2** In the navigation pane, choose **Edge Applications > Containerized Applications**. Then, click **Create Containerized Application** in the upper right corner.

**Step 3** Specify basic information.

- **Name:** name of a containerized application.
- **Instances:** number of instances of a containerized application. Only one instance is allowed.
- **Configuration Method**
  - **Custom:** Configure the application step by step. For details, see [Step 4 to Step 6](#).
  - **Application template:** Select a predefined application template and modify it based on your requirements. This method reduces repeated operations. The configurations in a template are the same as those set in [Step 4 to Step 6](#). For details on how to create a template, see [Application Templates](#).
- **Description:** description of the containerized application.
- **Tags:** Tags can be used to classify resources, facilitating resource management.

**Figure 3-26** Basic information

The screenshot displays a configuration form with the following elements:

- Service Instance:** A radio button selection with 'Professional Service Instance' selected.
- Creation Mode:** A dropdown menu set to 'Custom'.
- Name:** A text input field containing 'application'.
- Instances:** A text input field containing '1'.
- Configuration Method:** Two tabs, 'Custom' (active) and 'Application template'.
- Description:** A large text area with the placeholder 'Enter a description.' and a character count '0/255' at the bottom right.
- Tags:** Two input fields labeled 'Tag key' and 'Tag value'. Below them is the text 'You can add 20 more tags.'

**Step 4** Configure containers.

Click **Use Image** under the image to be used to deploy the application.

- **My Images:** Displays all the images you have created in [SWR](#).
- **Shared Images:** Displays images shared by other users. Shared images are managed and maintained in SWR. For details, see [Sharing a Private Image](#).

After selecting an image, specify container configurations.

- **Image Version:** Select the version of the image to be used to deploy the application.

**NOTICE**

Do not use version **latest** when deploying containers in the production environment. Because this will make it difficult to determine the version of the running image and to roll back the application properly.

- **Container Specifications:** Specify CPU, memory, Ascend AI accelerator card, and GPU quotas.
- **Ascend AI accelerator card:** The AI accelerator card configuration of the containerized application must be the same as that of the edge node actually deployed. Otherwise, the application will fail to be created. For details, see [how to configure the AI accelerator card during edge node registration](#).

**NOTE**

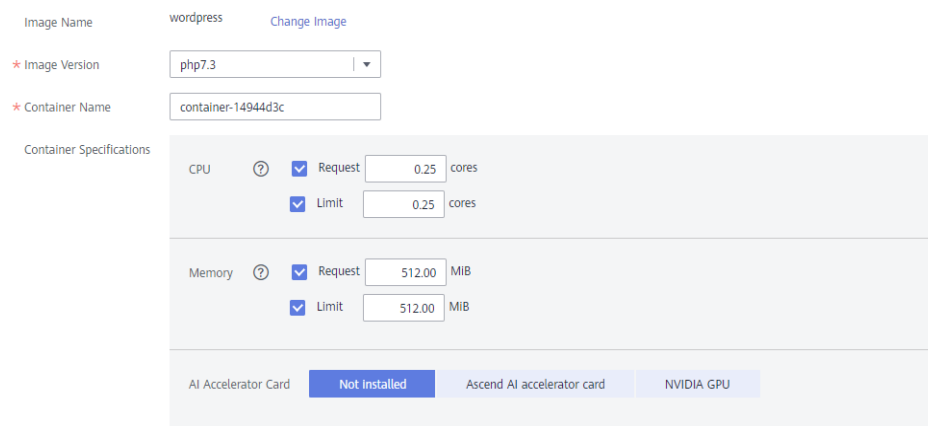
For NPUs after virtualization partition, only one virtualized NPU can be mounted to a container. The virtualized NPU can be allocated to another container only after the original container quits.

The following table lists the NPU types supported by Ascend AI accelerator cards.

**Table 3-8** NPU types

Type	Description
Ascend 310	Ascend 310 chips
Ascend 310B	Ascend 310B chips

**Figure 3-27** Container configuration



You can also configure the following advanced settings for the container:

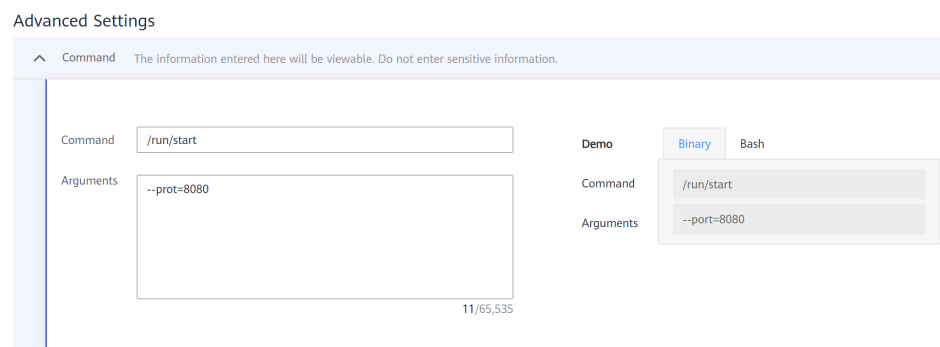
- **Command**  
A container image has metadata that stores image details. If lifecycle commands and arguments are not set, IEF runs the default commands and arguments provided during image creation, that is, **ENTRYPOINT** and **CMD** in the Dockerfile.

If the commands and arguments used to run a container are set during application creation, the default commands **ENTRYPOINT** and **CMD** are overwritten during image building. The rules are as follows:

**Table 3-9** Commands and arguments used to run a container

Image ENTRYPOINT	Image CMD	Command to Run a Container	Arguments to Run a Container	Command Executed
[touch]	[/root/test]	Not set	Not set	[touch /root/test]
[touch]	[/root/test]	[mkdir]	Not set	[mkdir]
[touch]	[/root/test]	Not set	[/opt/test]	[touch /opt/test]
[touch]	[/root/test]	[mkdir]	[/opt/test]	[mkdir /opt/test]

**Figure 3-28** Command



– **Command**

Enter an executable command, for example, **/run/start**.

If there are multiple commands, separate them with spaces. If the command contains a space, enclose the command in quotation marks ("").

**NOTE**

In the case of multiple commands, you are advised to run **/bin/sh** or other **shell** commands. Other commands are used as arguments.

– **Arguments**

Enter the argument that controls the container running command, for example, **--port=8080**.

If there are multiple arguments, separate them with line breaks.

• **Security Options**

- You can enable **Privileged Mode** to grant root permissions to the container for accessing host devices (such as GPUs and FPGAs).

– **RunAsUser Switch**

By default, IEF runs the container as the user defined during image building.

You can specify a user to run the container by turning this switch on, and entering the user ID (an integer ranging from 0 to 65534) in the text box displayed. If the OS of the image does not contain the specified user ID, the application fails to be started.

● **Environment Variables**

An environment variable affects the way a running container will behave. Variables can be modified after workload deployment. Currently, environment variables can be manually added, imported from secrets or ConfigMaps, or referenced from **hostIP**.

- **Added manually:** Customize a variable name and value.
- **Added from Secret:** You can customize a variable name. The variable value is referenced from secret configuration data. For details on how to create a secret, see [Secrets](#).
- **Added from ConfigMap:** You can customize a variable name. The variable value is referenced from ConfigMap configuration data. For details on how to create a ConfigMap, see [ConfigMaps](#).
- **Variable reference:** The variable value is referenced from **hostIP**, that is, the IP address of an edge node.

 **NOTE**

IEF does not encrypt the environment variables you entered. If the environment variables you attempt to configure contain sensitive information, you need to encrypt them before entering them and also need to decrypt them when using them.

IEF does not provide any encryption and decryption tools. If you need to configure cypher text, choose your own encryption and decryption tools.

● **Data Storage**

You can define a local volume and mount the local storage directory of the edge node to the container for persistent data storage.

Currently, the following four types of local volumes are supported:

- **hostPath:** used for mounting a host directory to the container. **hostPath** is a persistent volume. After an application is deleted, the data in **hostPath** still exists in the local disk directory of the edge node. If the application is re-created later, previously written data can still be read after the directory is mounted.

You can mount the application log directory to the **var/IEF/app/log/{appName}** directory of the host. In the directory name, **{appName}** indicates the application name. The edge node will upload the .log and .trace files in the **/var/IEF/app/log/{appName}** directory to AOM.

The mount directory is the log path of the application in the container. For example, the default log path of the Nginx application is **/var/log/nginx**. The permission must be set to **Read/Write**.

**Figure 3-29** Log volume mounting

Local Volume Name	Type	Mount Directory	Permission	Operation
log	hostPath	/var/IEF/app/log/ngi	/var/log/nginx	Read/Write Delete

- **emptyDir**: a simple empty directory used for storing transient data. It can be created in hard disks or memory. emptyDir is an empty directory after being mounted. The application can read files from and write files into the directory. emptyDir has the same lifecycle as the application. If the application is deleted, the data in emptyDir is deleted along with it.
- **configMap**: a type of resources that store configuration details required by the application. For details on how to create a ConfigMap, see [ConfigMaps](#).
- **secret**: a type of resources that store sensitive data, such as authentication, certificate, and key details. For details on how to create a secret, see [Secrets](#).

#### NOTICE

- The container path cannot be a system directory, such as / or /var/run. Otherwise, an exception occurs. You are advised to mount the container to an empty directory. If the directory is not empty, ensure that the directory does not contain any files that affect container startup. Otherwise, the files will be replaced, making it impossible for the container to be properly started. As a result, the application creation will fail.
- If the container is mounted into a high-risk directory, you are advised to use an account with minimum permissions to start the container. Otherwise, high-risk files on the host machine may be damaged.

- **Health Check**

Health check regularly checks the status of containers or workloads.

- **Liveness Probe**: The system executes the probe to check if a container is still alive, and restarts the instance if the probe fails. Currently, the system probes a container by HTTP request or command and determines whether the container is alive based on the response from the container.
- **Readiness Probe**: The system invokes the probe to determine whether the instance is ready. If the instance is not ready, the system does not forward requests to the instance.

For details, see [Health Check Configuration](#).

#### Step 5 Click Next.

Select a deployment object and specify an edge node.

**Figure 3-30** Specifying an edge node

\* Deployment Object

Edge Node

Advanced Settings

Restart Policy  Always restart  Restart upon failure  Do not restart  
The system restarts the container regardless of whether it had quit normally or abnormally.

Host PID  Disable  Enable  
Enabling this function allows instances to use the host's PID namespace. This function can be enabled only on edge nodes whose software version is 2.8.0 or later.



You can also configure the following advanced settings for the container:

- **Restart Policy**
  - **Always restart:** The system restarts the container regardless of whether it had quit normally or unexpectedly.
  - **Restart upon failure:** The system restarts the container only if it had previously quit unexpectedly.
  - **Do not restart:** The system does not restart the container regardless of whether it had quit normally or unexpectedly.

#### NOTICE

You are allowed to upgrade a containerized application and modify its access configuration only after you select **Always restart**.

- **Host PID**  
Enabling this function allows containers to share the PID namespace with the host where the edge node resides. In this way, you can perform container operations on the edge node, for example, starting and stopping a container. Similarly, you can perform operations related to the edge node in containers, for example, starting and stopping a process of the edge node.  
This function can only be enabled on edge nodes running software v2.8.0 or later.

#### Step 6 Click Next.

Container access supports bridged network and host network.

#### CAUTION

If the ports of containers deployed on the same edge node conflict, the containers will fail to be started.

- **Bridged network**  
The container uses an independent virtual network. A mapping between container and host ports needs to be configured to enable the external communication. After port mapping is configured, traffic destined for a host port is distributed to the mapping container port. For example, if container port 80 is mapped to host port 8080, the traffic destined for host port 8080 will be directed to container port 80.  
You can select a host NIC to enable the port bound to this NIC to communicate with the container port. Note that port mapping does not support NICs with IPv6 addresses.

**Figure 3-31** Bridged network

The screenshot shows the 'Container Network' configuration page. Under the 'Network' section, 'Bridged network' is selected. Below it, the 'Port Mapping' section contains a table with the following data:

Container Name	Container Port	Host Port
container-7...	80	Select fro... No NIC specified: 0.0.0.0 Use existi... 8080

There is an 'Add Port Mapping' button below the table.

- **Host network**

The network of the host (edge node) is used. To be specific, the container and the host use the same IP address, and network isolation is not required between them.

**Step 7** Click **Next** to confirm the specifications of the containerized application. If the specifications are correct, select **I have read and agree to the Huawei Cloud Service Level Agreement** and click **Create**.

----End

## Querying Application O&M Information

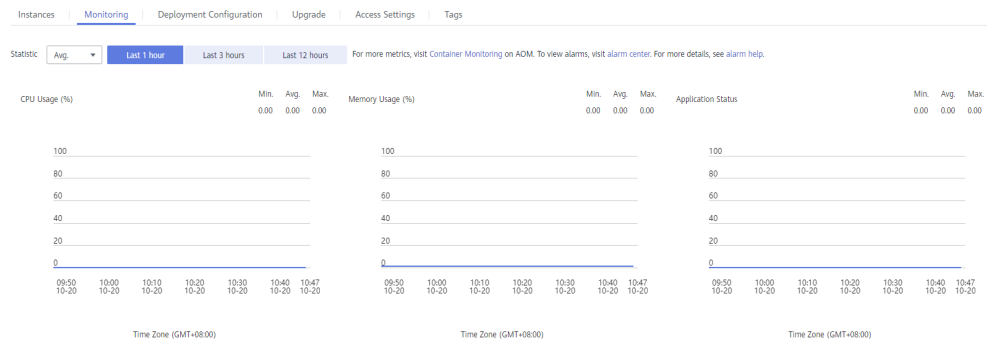
After an application is deployed, you can view the CPU and memory information of the application on the IEF console. You can also go to the **Container Monitoring** page on the AOM console to view additional metrics, or **Alarm Center** on the AOM console to add or view alarms.

**Step 1** Log in to the IEF console.

**Step 2** In the navigation pane, choose **Edge Applications > Containerized Applications**. Then, click an application name.

**Step 3** Click the **Monitoring** tab to view the monitoring details for the application.

**Figure 3-32** Application monitoring details



----End

## Upgrading an Application

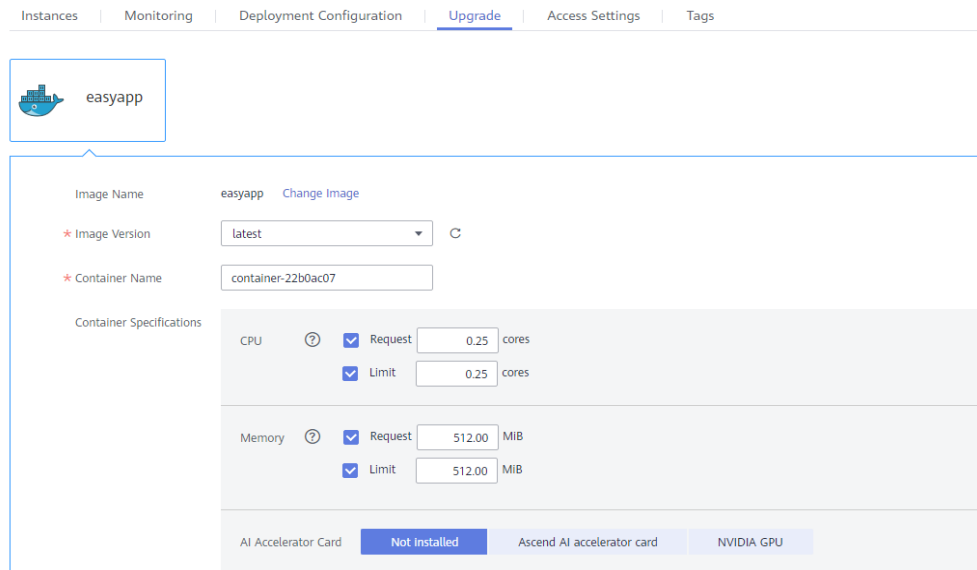
After an application is deployed, you can upgrade it. Rolling upgrades are used. That is, a new application instance is created first, and then the old application instance is deleted.

**Step 1** Log in to the IEF console.

**Step 2** In the navigation pane, choose **Edge Applications > Containerized Applications**. Then, click an application name.

**Step 3** Click the **Upgrade** tab and modify the container configurations. The parameter settings are the same as those in [Step 4](#).

**Figure 3-33** Upgrading an application



**Step 4** Click **Submit**.

----End

## Modifying Access Settings

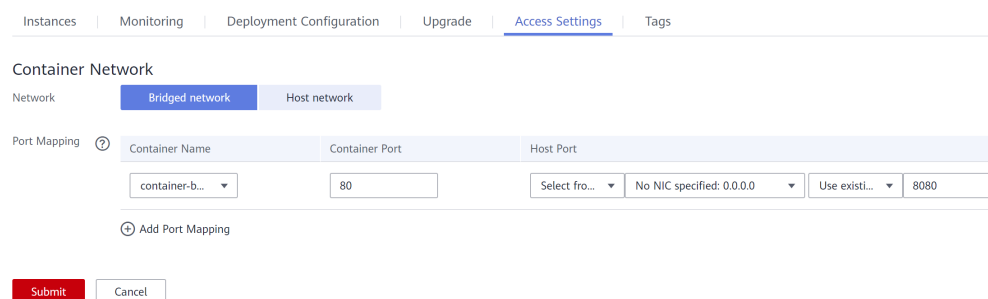
After an application is deployed, you can modify the access settings of the application.

**Step 1** Log in to the IEF console.

**Step 2** In the navigation pane, choose **Edge Applications > Containerized Applications**. Then, click an application name.

**Step 3** Click the **Access Settings** tab and modify the configurations. The parameter settings are the same as those in [Step 6](#).

**Figure 3-34** Modifying access settings



**Step 4** Click **Submit**.

----End

## 3.3.2 Application Templates

An application template defines the details required by containerized applications, such as the container image, configuration, disk mounting information, and resource usage.

Applications are created from an image. You need to create and upload an image to the image repository in advance.

### Creating an Application Template

**Step 1** Log in to the IEF console.

**Step 2** In the navigation pane, choose **Edge Applications > Application Templates**. Then, click **Create Application Template** in the upper right corner.

**Step 3** Specify basic information.

- **Name** (mandatory): name of an application template.
- **Version** (mandatory): version number of the application template.
- **Alias**: alias of the application template.
- **Architecture**: architecture supported by the application.
- **Description**: description of the application template.
- **Tags**: Tags can be used to classify application templates, facilitating template management. The tags specified in this step are only used to identify application templates. You can use these tags to filter the application templates during search.

**Step 4** Click **Next**. Then, add a container.

1. Click **Use Image** under the image to be used to deploy the application.
  - **My Images**: shows all the images you have uploaded to the image repository.
  - **Shared Images**: shows the images shared by other users.
2. Click **Next**. Then, specify container specifications.

**Figure 3-35** Image information

The screenshot displays the 'Add Container' configuration page. At the top, a progress bar indicates three steps: 1. Specify Basic Information (completed), 2. Add Container (current), and 3. Configure Advanced Settings. The main content area is divided into sections: 'Image Name' (nginx) with a 'Change Image' link, 'Image Version' (latest\_metadata) with a dropdown menu, and 'Container Specifications'. The specifications are organized into two rows: CPU and Memory. Each row has a 'Request' and 'Limit' field, both with checkboxes and units. CPU Request is 0.25 cores, Limit is 1.00 cores. Memory Request is 512.00 MIB, Limit is 512.00 MIB. At the bottom, there are three buttons: 'Not Installed' (highlighted in blue), 'Ascend AI accelerator card', and 'NVIDIA GPU'.

- **Image Version**: Select the version of the image to be used to deploy the application.

- **Container Specifications:** Specify CPU, memory, Ascend AI accelerator card, and GPU quotas.
- **Ascend AI accelerator card:** The AI accelerator card configuration of the containerized application must be the same as that of the edge node actually deployed. Otherwise, the application will fail to be created. For details, see [AI Accelerator Card Configurations](#).

 **NOTE**

For NPUs after virtualization partition, only one virtualized NPU can be mounted to a container. The virtualized NPU can be allocated to another container only after the original container quits.

The following table lists the NPU types supported by Ascend AI accelerator cards.

**Table 3-10** NPU types

Type	Description
Ascend 310	Ascend 310 chips
Ascend 310B	Ascend 310B chips

**Step 5** Click **Next**. Then, configure the application.

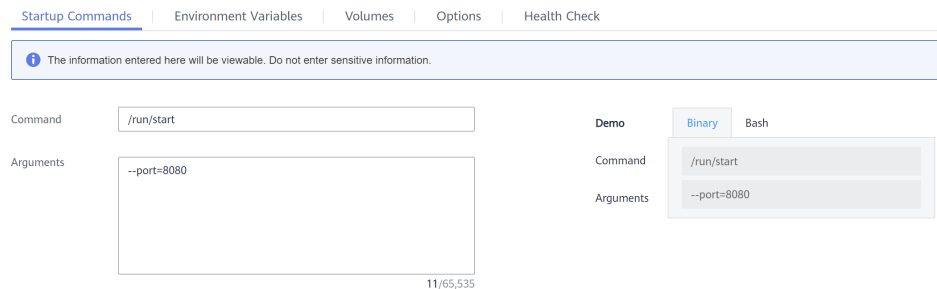
- **Startup Commands**

A container image has metadata that stores image details. If lifecycle commands and arguments are not set, IEF runs the default commands and arguments provided during image creation, that is, **ENTRYPOINT** and **CMD** in the Dockerfile.

If the commands and arguments used to run a container are set during application creation, the default commands **ENTRYPOINT** and **CMD** are overwritten during image building. The rules are as follows:

**Table 3-11** Commands and arguments used to run a container

Image ENTRYPOINT	Image CMD	Command to Run a Container	Arguments to Run a Container	Command Executed
[touch]	[/root/test]	Not set	Not set	[touch /root/test]
[touch]	[/root/test]	[mkdir]	Not set	[mkdir]
[touch]	[/root/test]	Not set	[/opt/test]	[touch /opt/test]
[touch]	[/root/test]	[mkdir]	[/opt/test]	[mkdir /opt/test]

**Figure 3-36** Startup command

### – Command

Enter an executable command, for example, **/run/start**.

If there are multiple commands, separate them with spaces. If the command contains a space, enclose the command in quotation marks ("").

#### NOTE

In the case of multiple commands, you are advised to run **/bin/sh** or other **shell** commands. Other commands are used as arguments.

### – Arguments

Enter the argument that controls the container running command, for example, **--port=8080**.

If there are multiple arguments, separate them with line breaks.

## ● Environment Variables

An environment variable affects the way a running container will behave. You can modify environment variables even after applications are deployed.

Click **Add Environment Variable** to add a variable. Currently, environment variables can be manually added, imported from secrets or ConfigMaps, or referenced from **hostIP**.

- **Added manually:** Customize a variable name and value.
- **Added from Secret:** You can customize a variable name. The variable value is referenced from secret configuration data. For details on how to create a secret, see [Secrets](#).
- **Added from ConfigMap:** You can customize a variable name. The variable value is referenced from ConfigMap configuration data. For details on how to create a ConfigMap, see [ConfigMaps](#).
- **Variable reference:** The variable value is referenced from **hostIP**, that is, the IP address of an edge node.

#### NOTE

IEF does not encrypt the environment variables you entered. If the environment variables you attempt to configure contain sensitive information, you need to encrypt them before entering them and also need to decrypt them when using them.

IEF does not provide any encryption and decryption tools. If you need to configure cypher text, choose your own encryption and decryption tools.

## ● Volumes

A volume is used for storage when the container is running. Currently, the following four volume types are supported:

- **hostPath**: used for mounting a host directory to the container. hostPath is a persistent volume. After an application is deleted, the data in hostPath still exists in the local disk directory of the edge node. If the application is re-created later, previously written data can still be read after the directory is mounted.

You can mount the application log directory to the **var/IEF/app/log/{appName}** directory of the host. In the directory name, **{appName}** indicates the application name. The edge node will upload the .log and .trace files in the **/var/IEF/app/log/log/{appName}** directory to AOM.

**Figure 3-37** Log volume mounting

Local Volume Name	Type	Mount Directory	Permission	Operation
log	hostPath	/var/IEF/app/log/applicat	/home/log	Read only

⊕ Add Volume

- **emptyDir**: a simple empty directory used for storing transient data. It can be created in hard disks or memory. emptyDir is an empty directory after being mounted. The application can read files from and write files into the directory. emptyDir has the same lifecycle as the application. If the application is deleted, the data in emptyDir is deleted along with it.
- **configMap**: a type of resources that store configuration details required by the application. For details on how to create a ConfigMap, see [ConfigMaps](#).
- **secret**: a type of resources that store sensitive data, such as authentication, certificate, and key details. For details on how to create a secret, see [Secrets](#).

## NOTICE

- The container path cannot be a system directory, such as **/** or **/var/run**. Otherwise, an exception occurs. You are advised to mount the container to an empty directory. If the directory is not empty, ensure that the directory does not contain any files that affect container startup. Otherwise, the files will be replaced, making it impossible for the container to be properly started. As a result, the application creation will fail.
- If the container is mounted into a high-risk directory, you are advised to use an account with minimum permissions to start the container. Otherwise, high-risk files on the host machine may be damaged.

## Options

- **Privileged Mode**

You can enable **Privileged Mode** to grant root permissions to the container for accessing host devices (such as GPUs and FPGAs).

- **RunAsUser Switch**

By default, IEF runs the container as the user defined during image building.

You can specify a user to run the container by turning this switch on, and entering the user ID (an integer ranging from 0 to 65534) in the text box

displayed. If the OS of the image does not contain the specified user ID, the application fails to be started.

- **Restart Policy**

**Always restart:** The system restarts the container regardless of whether it had quit normally or unexpectedly.

**Restart upon failure:** The system restarts the container only if it had previously quit unexpectedly.

**Do not restart:** The system does not restart the container regardless of whether it had quit normally or unexpectedly.

- **Container Network**

**Host network:** The network of the host (edge node) is used. To be specific, the container and the host use the same IP address, and network isolation is not required between them.

**Bridged network:** The container uses an independent virtual network. A mapping between container and host ports needs to be configured to enable the external communication. After port mapping is configured, traffic destined for a host port is distributed to the mapping container port. For example, if container port 80 is mapped to host port 8080, the traffic destined for host port 8080 will be directed to container port 80.

- **Health Check**

- **Liveness Probe:** The system executes the probe to check if a container is still alive, and restarts the instance if the probe fails. Currently, the system probes a container by HTTP request or command and determines whether the container is alive based on the response from the container.

- **Readiness Probe:** The system invokes the probe to determine whether the instance is ready. If the instance is not ready, the system does not forward requests to the instance.

For details, see [Health Check Configuration](#).

**Step 6** After configuration, click **Create**.

----End

## Creating an Application Template Version

You can create multiple application versions for an application template to facilitate application management.

**Step 1** Log in to the IEF console.

**Step 2** In the navigation pane, choose **Edge Applications > Application Templates**.

**Step 3** Click the application template whose version is to be added. The application template details page is displayed.

**Step 4** In the lower left corner of the page, click **Create Application Version**.



**Figure 3-38** Creating an application version

Version	Image Name	Image Version
v1	nginx	latest

**Step 5** Enter a version, and click **Next**. Then, configure other parameters by following the description in [Creating an Application Template](#).

----End

### 3.3.3 ConfigMaps

ConfigMaps store configuration details required for workloads. ConfigMaps decouple configuration files from container images to enhance workload portability.

ConfigMaps allow you to:

- Manage configurations for different environments and services.
- Deploy workloads in different environments. Multiple versions are supported for configuration files so that you can update and roll back workloads easily.
- Quickly import configurations in the form of files to containers.

### Creating a ConfigMap

**Step 1** Log in to the IEF console.

**Step 2** In the navigation pane, choose **Edge Applications > Application Configuration**. Then, click **Create ConfigMap** in the upper right corner.

**Step 3** Specify the ConfigMap details.

**Figure 3-39** Creating a ConfigMap

★ Name

Data ⓘ The information entered here will be viewable. Do not enter sensitive information.

Key	Value	Operation
<input type="text" value="key"/>	<input type="text" value="value"/>	Delete

⊕ Add Data

Description

0/255

- **Name:** Enter a ConfigMap name.
- **Data:** The configuration data is key-value pairs. Enter a property name and value.

**Step 4** Click **Create**. After the ConfigMap is created, the ConfigMap list page is displayed.

----End

## Using a ConfigMap

You can use ConfigMaps to configure data storage in the advanced settings when creating a containerized application.

**Figure 3-40** Using a ConfigMap

The screenshot shows a configuration page for 'Data Storage'. At the top, there is a note: 'Note: The /etc/localtime file is automatically mounted by default. Do not mount it again. Otherwise, the application creation will fail.' Below this is a table with columns: 'Local Volume Name', 'Type', 'Mount Directory', 'Permission', and 'Operation'. A row is shown with 'key' in the 'Local Volume Name' field, 'configMap' in the 'Type' dropdown, 'new-config' in the 'Mount Directory' dropdown, '/tmp0' in the 'Mount Directory' text field, 'Read o...' in the 'Permission' dropdown, and '420' in the 'Permission' text field. There is a 'Delete' button next to the row. At the bottom, there is a '+ Add Volume' button.

After the ConfigMap is mounted to the container, files are created in the mount directory based on the ConfigMap content. Each property name and value pair is generated as a file. The property name is the file name, and the value is the content of the file. For instance, a ConfigMap could have the property name set to **key** and the property value set to **value**. After the ConfigMap is mounted to the **/tmp0** directory, a file named **key** is generated in the **/tmp0** directory and the file content is **value**.

### 3.3.4 Secrets

Secrets store sensitive, user-defined information such as authentication details, certificates, and keys, and can then be loaded to containerized applications. For example, you can mount a volume of the secret type for data store to make it a file in the container, or load a secret to make it an environment variable in the container.

#### NOTICE

Secrets may involve sensitive user information. If the secrets you attempt to configure contain sensitive information, you need to encrypt them before entering them and also need to decrypt them through applications.

## Creating a Secret

**Step 1** Log in to the IEF console.

**Step 2** In the navigation pane, choose **Edge Applications > Application Configuration**. Then, click **Create Secret** in the upper right corner.

**Step 3** Specify the secret details.

**Figure 3-41** Creating a secret

- **Name:** name of the secret.
- **Type:** **Opaque** secrets are supported. Data is in the key-value pair format and Base64-encoded. For details, see [Base64 Encoding](#).
- **Data:** Key-value pairs. Enter a property name and value. The value must be Base64-encoded.
- **Description:** description of the secret.

**Step 4** Click **Create**. After the secret is created, the secret list page is displayed.

----End

## Base64 Encoding

To perform Base64 encoding on a character string, run the `echo -n {Content to be encoded} | base64` command. For example:

```
root@ubuntu:~# echo -n "example value" | base64
ZXhhbXBsZSB2YWx1ZQ==
```

## Using a Secret

You can use secrets to configure data storage in the advanced settings when creating a containerized application.

**Figure 3-42** Using a secret

After the secret is mounted to the container, files are created in the mount directory based on the secret content. Each property name and value pair is

generated as a file. The property name is the file name, and the value is the content of the file. For instance, a secret could have the property name set to **key** and the property value set to **ZXhhbXBsZSB2YWx1ZQ==**. After the secret is mounted to the **/tmp0** directory, a file named **key** is generated in the **/tmp0** directory and the file content is **example value**.

### 3.3.5 Encryption Data

Encryption data is used to store and encrypt sensitive information. Edge applications can access plaintext data through MQTT server.

#### Creating Encryption Data

**Step 1** Log in to the IEF console.

**Step 2** In the navigation pane, choose **Edge Applications > Application Configuration**. Then, click **Create Encryption Data** in the upper right corner. On the page that is displayed, specify the parameters.

**Figure 3-43** Creating encryption data

Service Instance: Professional Service Instance

\* Name: data

\* Encryption Items: At least one item must be added. The information entered here will be viewable. Do not enter sensitive information.

Key	Value	Operation
key	.....	Delete
	.....	

+ Add Encryption Item

Description: Enter a description. (0/255)

The encryption data is stored in key-value pairs. The value needs to be entered twice. Multiple encryption items can be added to each encryption data record.

**Step 3** Click **Create**.

You can view encryption data after creating it. However, for data privacy protection, data cannot be viewed in plaintext. You can also edit or delete encryption data.

**Figure 3-44** Encryption data

Encryption Items | Bound Nodes

+ Add: At least one item must exist.

Key	Value	Operation
password	.....	Edit Delete

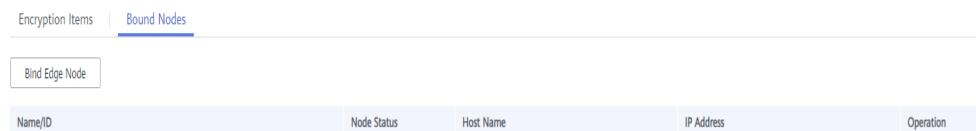
----End

## Binding Encryption Data to Edge Nodes

Encryption data can be bound to edge nodes that are not in the running state. When an edge node is restored to **Running**, the bound data will be automatically synchronized to the node. You can bind encryption data to an edge node in either of the following ways:

- On the encryption data details page, click **Bind Node** and then **Bind Edge Node**.

**Figure 3-45** Binding an edge node



- On the edge node details page, click the **Configuration** tab. In the **Encryption Data** area, click **Bind**.

**Figure 3-46** Binding encryption data to an edge node



## Using Encryption Data

After encryption data is bound to an edge node, you can obtain the data by using the MQTT client on the edge node.

A certificate must be used for security authentication when encryption data is requested. For details about the authentication method, see [Performing Security Authentication Using Certificate](#).

**Step 1** Subscribe to the topic described in [Encryption Data Acquisition](#).

**Topic:** \$hw/{project\_id}/encryptdatas/{encryptdata\_name}/properties/{properties\_name}/plaintext

**Step 2** Publish the topic described in [Encryption Data Request](#).

**Topic:** \$hw/{project\_id}/encryptdatas/{encryptdata\_name}/properties/{properties\_name}/decrypt

After the request is published, the decrypted data is sent to the topic subscribed in [Step 1](#).

----End

### 3.3.6 Health Check Configuration

Health check regularly checks the status of containers or workloads.

- **Liveness Probe:** The system executes the probe to check if a container is still alive, and restarts the instance if the probe fails. Currently, the system probes

a container by HTTP request or command and determines whether the container is alive based on the response from the container.

- **Readiness Probe:** The system invokes the probe to determine whether the instance is ready. If the instance is not ready, the system does not forward requests to the instance.

IEF can perform a health check by HTTP request or through a CLI.

## HTTP Request-based Check

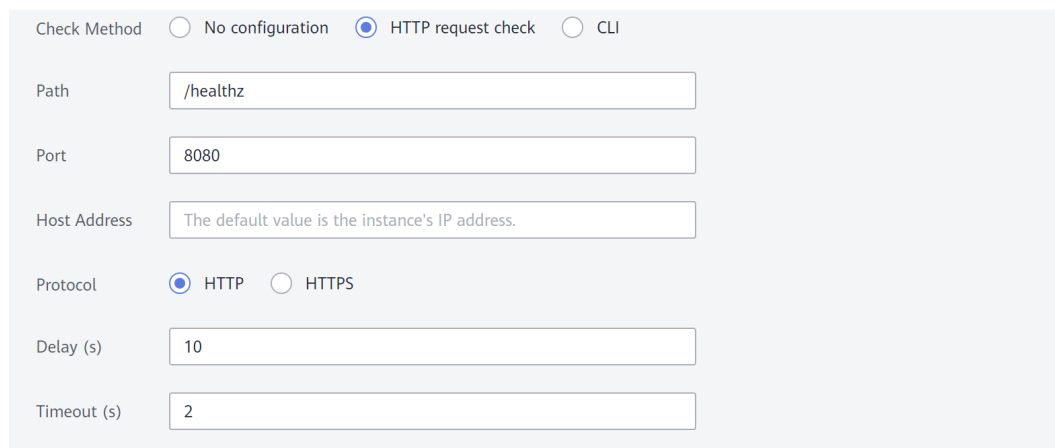
IEF periodically initiates an HTTP GET request to a container. If HTTP code 2xx or 3xx is received, the container is healthy.

For example, if health check parameters are set as shown in the following figure, IEF will send an HTTP GET request to **http://{Instance's IP address}/healthz:8080** 10 seconds after the container starts. If no response is received within 2 seconds, the health check fails. If status code 2xx or 3xx is received, the container is healthy.

### NOTE

You do not need to specify the host address. By default, the instance's IP address is used (that is, requests are sent to the container) unless you have special requirements.

**Figure 3-47** HTTP request-based check

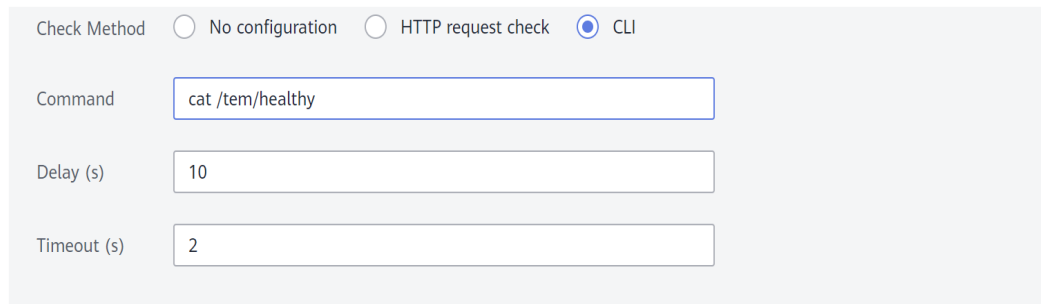


Check Method	<input type="radio"/> No configuration	<input checked="" type="radio"/> HTTP request check	<input type="radio"/> CLI
Path	<input type="text" value="/healthz"/>		
Port	<input type="text" value="8080"/>		
Host Address	<input type="text" value="The default value is the instance's IP address."/>		
Protocol	<input checked="" type="radio"/> HTTP	<input type="radio"/> HTTPS	
Delay (s)	<input type="text" value="10"/>		
Timeout (s)	<input type="text" value="2"/>		

## CLI-based Check

The probe runs commands in the container and checks the command output. If the command output is 0, the container is healthy.

For example, if health check parameters are set as shown in the following figure, IEF will run **cat /tmp/healthy** 10 seconds after the container starts. If no response is received within 2 seconds, the health check fails. If the command output is 0, the container is healthy.

**Figure 3-48** CLI-based check

Check Method  No configuration  HTTP request check  CLI

Command

Delay (s)

Timeout (s)

## 3.4 Edge-Cloud Messages

### 3.4.1 Edge-Cloud Message Overview

IEF provides the function of routing messages exchanged between the edge and cloud. Based on configured routes, IEF forwards messages to the corresponding endpoint. In this way, messages can be forwarded based on specified paths, enhancing flexibility in data routing control and improves data security.

- Message endpoint: a party that sends or receives a message. It can be an end device or a cloud service.
- Message route: a message forwarding path.

### Message Endpoints

IEF provides the following default message endpoints:

- **SystemEventBus**: MQTT broker on an edge node, which can communicate with other endpoints on behalf of the edge node. It can function as a source endpoint to send data to the cloud, or as a destination endpoint to receive messages from the cloud. MQTT topics on the edge node are used as endpoint resources of the MQTT broker.
- **SystemREST**: a REST gateway interface in the cloud. It can function as a source endpoint to send REST requests to the edge. REST request paths are used as endpoint resources of SystemREST.

You can also create the following message endpoints:

- **Service Bus**: an endpoint deployed on an edge node to process transaction requests. It can function as a destination endpoint to process file upload requests. REST request paths are used as endpoint resources of Service Bus.
- **DIS**: data injection service. It can function as a destination endpoint to receive data forwarded by IEF. DIS streams created in DIS are used as endpoint resources.
- **API Gateway**: API gateway service. It can function as a destination endpoint to receive data forwarded by IEF. API URLs created in API Gateway are used as endpoint resources.

## Routes

Currently, IEF supports the following message forwarding paths:

- **SystemREST -> Service Bus:** The REST gateway interface in the cloud is called to obtain file services on edge nodes.
- **SystemREST -> SystemEventBus:** The REST gateway interface in the cloud is called to send messages to SystemEventBus (MQTT broker) on edge nodes.
- **SystemEventBus -> DIS/API Gateway:** You can publish end device data to a custom topic in the MQTT broker of an edge node. IEF then forwards the device data to a DIS stream or an API Gateway address. Then, you can extract the data for processing and analysis. You must customize an MQTT topic when creating the route. For details about custom topics, see [Custom Topics](#).

To use the message routing function, [create endpoints](#) and then [create routes](#).

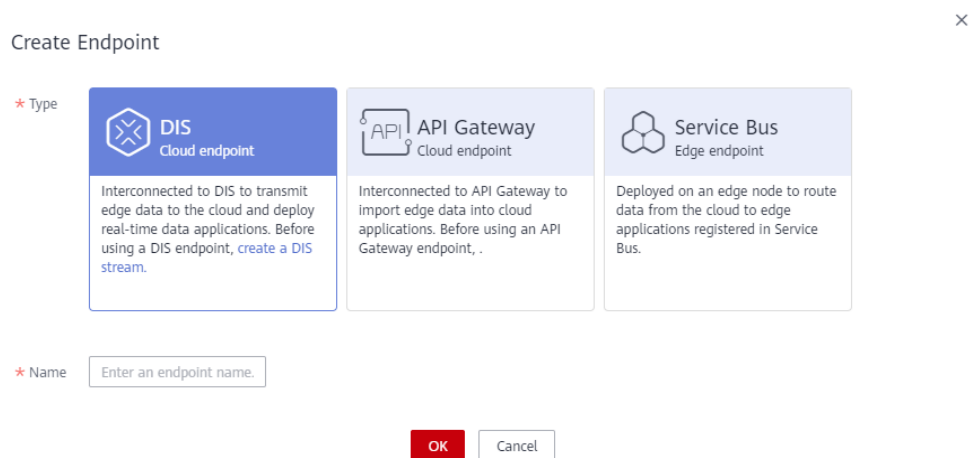
## Creating an Endpoint

**Step 1** Log in to the IEF console.

**Step 2** In the navigation pane, choose **Edge-Cloud Messages > Endpoints**.

**Step 3** Click **Create Endpoint** in the upper right corner, and set the endpoint parameters.

**Figure 3-49** Creating an endpoint



- **Type:** Select an endpoint type. **DIS**, **API Gateway**, and **Service Bus** are supported.
- **Name:** Enter an endpoint name.
- **Service Port:** This parameter is available only for endpoints of the **Service Bus** type. It ranges from 1 to 65535.

**Step 4** Click **OK**. The endpoint is successfully created and the endpoint list page is displayed.

----End



## Creating a Route

**Step 1** Log in to the IEF console.

**Step 2** In the navigation pane, choose **Edge-Cloud Messages > Routes**.

**Step 3** Click **Create Route** in the upper right corner.

**Step 4** Set the route parameters.

**Figure 3-50** Creating a route

Service Instance Professional Service Instance

\* Name

\* Source Endpoint SystemREST

\* Source Endpoint Resource

\* Destination Endpoint SystemEventBus

\* Destination Endpoint Resource

Description

0/255

- **Name:** Enter a route name.

---

 **CAUTION**

Message routes and **system subscriptions** are of the same resource type. Their names cannot conflict with each other.

- **Source Endpoint:** Select a source endpoint, for example, **SystemREST**.
- **Source Endpoint Resource:** Select a source endpoint resource.
- **Destination Endpoint:** Select a destination endpoint, for example, **SystemEventBus**.
- **Destination Endpoint Resource:** Select a destination endpoint resource.

**Step 5** Click **Create**. The route is successfully created and the route list page is displayed.

IEF forwards messages sent to the specified resource of the source endpoint to the specified resource of the destination endpoint based on the route.

----End

## Disabling or Enabling a Route

- To disable a route, click **Disable** in the row where the route resides.  
After the route is disabled, IEF will not forward messages destined for the specified resource of the source endpoint.
- To enable a route, click **Enable** in the row where the route resides.  
After the route is enabled, IEF will forward messages destined for the specified resource of the source endpoint.

## 3.4.2 Cloud Delivering Messages to Edge Nodes

### Scenario

IEF can deliver messages from SystemREST on the cloud to SystemEventBus (MQTT broker) or ServiceBus on edge nodes. This section uses SystemEventBus as an example.

To be specific, IEF calls the open REST gateway interface to send messages to SystemEventBus on an edge node based on the node ID and custom topic. Your end devices can receive messages from the cloud by subscribing to custom topics.

The procedure is as follows:

1. [Creating a Route](#)
2. [Sending a Message](#)
3. [Receiving a Message](#)

### Creating a Route

SystemREST and SystemEventBus are default endpoints and do not need to be created. SystemREST indicates the REST gateway interface of IEF in the region. SystemEventBus indicates the MQTT broker of an edge node.

- Step 1** Log in to the IEF console.
- Step 2** In the navigation pane, choose **Edge-Cloud Messages > Routes**.
- Step 3** Click **Create Route** in the upper right corner.
- Step 4** Set parameters.

**Figure 3-51** Creating a route

* Name	<input type="text" value="systemrest2systemeventbus"/>
* Source Endpoint	<input type="text" value="SystemREST"/>
* Source Endpoint Resource	<input type="text" value="/"/>
* Destination Endpoint	<input type="text" value="SystemEventBus"/>
* Destination Endpoint Resource	<input type="text" value="/"/>
Description	<div style="border: 1px solid #ccc; padding: 5px; min-height: 100px;">Enter a description.</div> 0/255

- **Name:** Enter a route name, for example, **SystemREST2SystemEventBus**.

---

 **CAUTION**

Message routes and **system subscriptions** are of the same resource type. Their names cannot conflict with each other.

---

- **Source Endpoint:** Select **SystemREST**.
- **Source Endpoint Resource:** Enter a URL path starting with a slash (/). You can use {} for fuzzy match. For example, **/aaa/{any-str}/bbb** can match **/aaa/cc/bbb** or **/aaa/dd/bbb**. The specified source resource is used as a matching field for calling the REST interface.
- **Destination Endpoint:** Select **SystemEventBus**.
- **Destination Endpoint Resource:** Enter the topic name prefix used when the message is forwarded to the MQTT broker.

 **NOTE**

If both **Source Endpoint Resource** and **Destination Endpoint Resource** are set to /, IEF will forward all messages sent to the REST interface to the MQTT broker of the corresponding edge nodes. The topic name contained in the messages is the same as the request URL.

**Step 5** Click **Create**.

The created route is displayed in the message route list.

**Figure 3-52 Routes**

Route Name	Source Endpoint	Destination Endpoint	Status	Forwarded Messages
SystemREST2SystemEv...	SystemREST Cloud	SystemEventBus Edge	Enabled	Total: 0 Successful: 0 Failed: 0

----End

## Sending a Message

To deliver a message from SystemREST in the cloud to SystemEventBus on an edge node, obtain the SystemREST endpoint property, construct a request, and send the request to SystemEventBus through a route.

### Step 1 Obtain the SystemREST endpoint property.

1. Log in to the IEF console.
2. In the navigation pane, choose **Edge-Cloud Messages > Endpoints**. In the row where SystemREST is located, the value of **public** in the **Property** column is the endpoint of SystemREST, as shown in the following figure.

**Figure 3-53 SystemREST endpoint property**

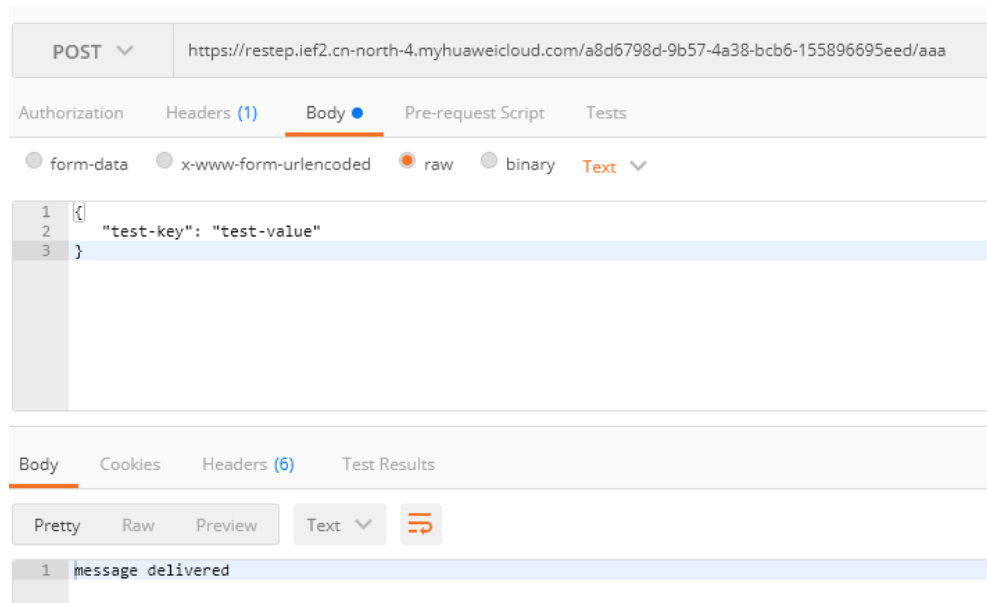
Name	Type	Position	Property
SystemEventBus	eventbus (Default)	Edge	
SystemREST	rest (Default)	Cloud	public = restep.ief2.cn-north-4.myhuaweicloud.com

### Step 2 Construct a request and send it.

An HTTPS request needs to be constructed and sent to SystemREST. The details are as follows:

- Method: POST
- URL: **https://{rest\_endpoint}/{node\_id}/{topic}**. In the URL, *{rest\_endpoint}* is the endpoint obtained in [Step 1](#), *{node\_id}* is the edge node ID, and *{topic}* is the message topic, that is, the source endpoint resource defined in [Creating a Route](#).
- Body: content of the message to be sent, which is user-defined.
- Header: **X-Auth-Token**, which is a valid token obtained from IAM. For details on how to obtain a token, see [Obtaining a User Token](#).

For example, use Postman to send the following message:

**Figure 3-54** Sending a message

----End

## Receiving a Message

**Step 1** Log in to the edge node.

**Step 2** Use the MQTT client to receive messages.

Currently, edge nodes support two types of MQTT brokers.

- Built-in MQTT broker (using port 8883): To use this type of MQTT broker, edge nodes must pass certificate authentication and subscribe to corresponding topics. For details about certificate authentication, see [Performing Security Authentication Using Certificate](#).
- External MQTT broker (using port 1883): To use this type of MQTT broker, install a third-party MQTT broker on edge nodes and enable edge nodes to subscribe to corresponding topics.

This section describes how to use an external MQTT broker, for example, Mosquitto, to receive messages. The procedure is as follows:

- In the Ubuntu OS, run the following commands to install Mosquitto:

```
apt-get install mosquitto
systemctl start mosquitto
systemctl enable mosquitto
```
- In the CentOS OS, run the following commands to install Mosquitto:

```
yum install epel-release
yum install mosquitto
systemctl start mosquitto
systemctl enable mosquitto
```

After the installation is complete, run the topic subscription command. If a message is sent after the subscription, the edge node will receive the message. In the following command, # indicates that any topic is subscribed to. You can replace # with a specified topic, for example, /aaa or /bbb.

```
[root@ief-node ~]# mosquitto_sub -t '#' -d
Client mosq-m02iwj4j2ISMw6rw sending CONNECT
```

```
Client mosq-m02iwj4j2ISMw6rw received CONNACK (0)
Client mosq-m02iwj4j2ISMw6rw sending SUBSCRIBE (Mid: 1, Topic: #, QoS: 0, Options: 0x00)
Client mosq-m02iwj4j2ISMw6rw received SUBACK
Subscribed (mid: 1): 0
Client mosq-m02iwj4j2ISMw6rw received PUBLISH (d0, q0, rQ, mQ, '/aaa', ... (31 bytes))
{
  "test-key": "test-value"
}
```

----End

### 3.4.3 Edge Nodes Reporting Messages to the Cloud

#### Scenario

IEF allows edge nodes to report messages to the cloud.

You can publish data to a custom topic in SystemEventBus (MQTT broker) of edge nodes. IEF forwards the data to a DIS stream or an API Gateway address. Then, you can extract the data for processing and analysis.

This section uses the DIS endpoint as an example. The method of using the API Gateway endpoint is similar. The procedure is as follows:

1. [Creating an Endpoint](#)
2. [Buying a DIS Stream](#)
3. [Creating a Route](#)
4. [Sending a Message](#)

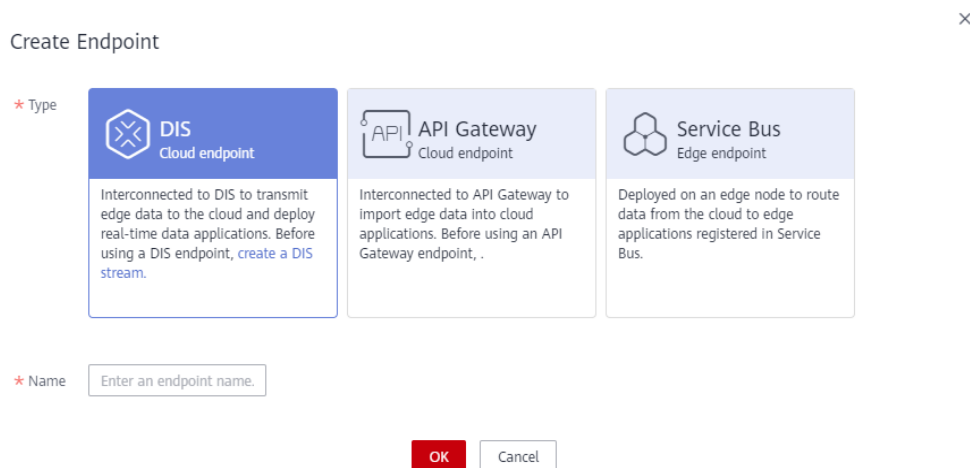
#### Creating an Endpoint

**Step 1** Log in to the IEF console.

**Step 2** In the navigation pane, choose **Edge-Cloud Messages > Endpoints**.

**Step 3** Click **Create Endpoint** in the upper right corner. Select **DIS** for **Type**, and enter an endpoint name.

**Figure 3-55** Creating an endpoint



**Step 4** Click **OK**.

----End

## Buying a DIS Stream

Before sending messages to DIS, buy a DIS stream.

**Step 1** Log in to the DIS console.

**Step 2** Click Buy Stream in the right corner and set parameters.

**Figure 3-56** Buying a DIS stream

The screenshot shows the configuration page for buying a DIS stream. It is divided into several sections:

- Billing Mode:** A dropdown menu set to "Pay-per-use".
- Region:** A dropdown menu set to "CN East-Shanghai1". Below it, a note states: "Regions are geographic areas isolated from each other. Resources are region-specific and cannot be used across regions through internal network connections. For low network latency and quick resource access, select the nearest region."
- Stream Name:** A text input field containing "dis-ief-1". Below it, a note states: "The system automatically populates an editable stream name that contains the prefix 'dis' followed by four alphanumeric characters."
- Stream Type:** Two tabs: "Common" (selected) and "Advanced". A help icon is next to the "Advanced" tab.
- Partitions:** A numeric input field set to "1", with minus and plus buttons. A "Partition Calculator" button is next to it. A note says: "You can use a maximum of 49 partitions. Learn how to increase quota." Below this, it says: "Selected: Common DIS stream | 1 partition | maximum stream capacity: 1 MB/s (write); 2 MB/s (read)".
- Data Retention (hours):** A numeric input field set to "24", with minus and plus buttons.
- Source Data Type:** Three tabs: "BLOB" (selected), "JSON", and "CSV". A help icon is next to the "CSV" tab.
- Auto Scaling:** A toggle switch that is currently turned off.
- Enterprise Project:** A dropdown menu set to "default" with a refresh icon.
- Advanced Settings:** Two buttons: "Skip" (selected) and "Configure".

**Step 3** Click **Next**, confirm the product specifications, and click **Submit**.

----End

## Creating a Route

**Step 1** Log in to the IEF console.

**Step 2** In the navigation pane, choose **Edge-Cloud Messages > Routes**.

**Step 3** Click **Create Route** in the upper right corner.

**Step 4** Set parameters.

**Figure 3-57** Creating a route

- **Name:** Enter a route name.

**CAUTION**

Message routes and **system subscriptions** are of the same resource type. Their names cannot conflict with each other.

- **Source Endpoint:** Select **SystemEventBus**.
- **Source Endpoint Resource:** Select **Custom topic**, select the edge node that will send messages, and enter a topic name.
- **Destination Endpoint:** Select the endpoint created in **Creating an Endpoint**.
- **Destination Endpoint Resource:** Select the DIS stream purchased in **Buying a DIS Stream**.

**NOTE**

- Record the topic, as shown in the red box in the preceding figure. After the route is created, you can view the topic in the **Source Endpoint** column of the route list.
- After customizing a topic, you need to use the complete topic (marked in red in the figure above) for messaging.

**Step 5** Click **Create**.

----End

**Sending a Message**

Use the MQTT client on the edge node to send messages.

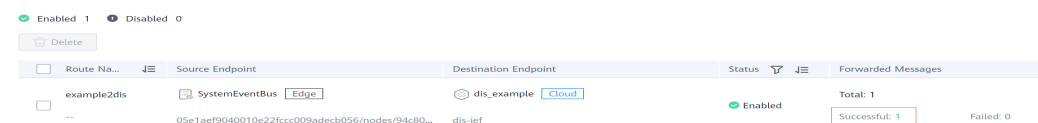


In this example, the message needs to be published to the topic specified in [Creating a Route](#). As shown in the following figure, `mosquitto_pub` is used to send a message.

```
[root@ief-node ~]# mosquitto_pub -t '05e1aef9040010e22fcc009adecb056/nodes/7092ad14-adee-4a09-b969-1505bbdecef5/user/aaa' -d -m '{ "edgemsg": "msgToCloud"}'
Client mosq-p5LouPQIW2gx0JPkRF sending CONNECT
Client mosq-p5LouPQIW2gx0JPkRF received CONNACK (0)
Client mosq-p5LouPQIW2gx0JPkRF sending PUBLISH (d0, q0, r0, m1, '05e1aef9040010e22fcc009adecb056/nodes/7092ad14-adee-4a09-b969-1505bbdecef5/user/aaa', ... (26 bytes))
Client mosq-p5LouPQIW2gx0JPkRF sending DISCONNECT
```

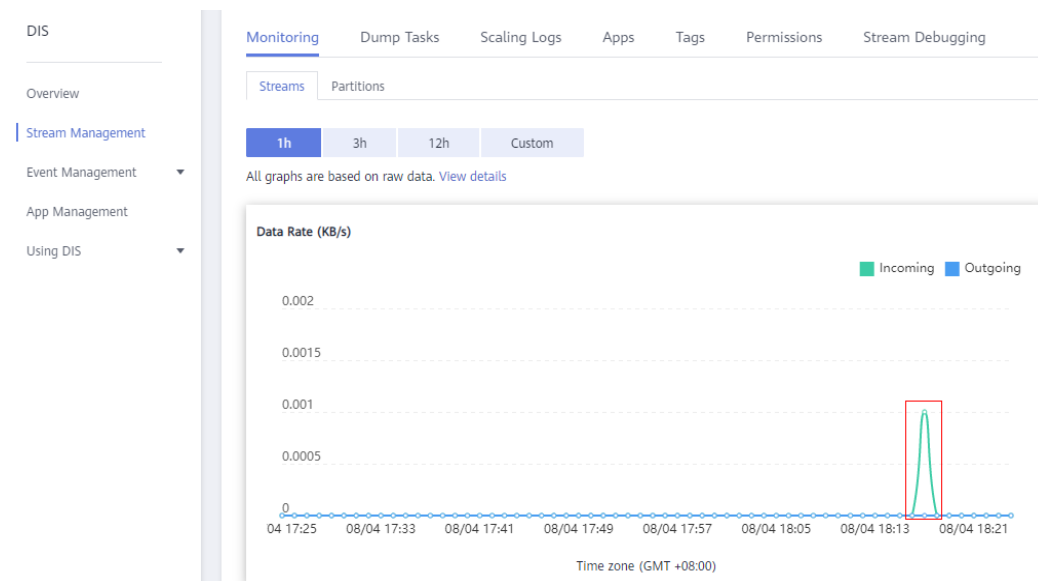
After the message is sent, you can view the **Forwarded Messages** column in the route list to check whether the message is successfully forwarded by the route.

**Figure 3-58** Number of forwarded messages



You can view incoming messages on the DIS console.

**Figure 3-59** DIS data monitoring



## Obtaining Data

After data is forwarded to DIS stream, you can extract the data for processing and analysis. For details on how to obtain data from DIS, see [Obtaining Data from DIS](#).

## 3.4.4 System Subscriptions

### Scenario

IEF provides the system subscription function for you to subscribe to IEF resource change events. When a resource is created, updated, or deleted, IEF will send a

message to the specified API Gateway endpoint so that you can obtain resource changes in a timely manner.

System subscription is the special implementation of edge-cloud messages. IEF sends event messages about specific resources to a specified topic and calls an API of API Gateway to notify you of resource changes.

## Events That Can Be Subscribed To

You can subscribe to the following events from IEF.

**Table 3-12** Events that can be subscribed to

System Event	Topic	Resource Type	Operation
Instance creation	\$hw/events/instance/+/created	Application instance	Create
Instance update	\$hw/events/instance/+/updated	Application instance	Update
Instance deletion	\$hw/events/instance/+/deleted	Application instance	Delete
Application deletion	\$hw/events/deployment/+/deleted	Containerized application	Delete
Application creation	\$hw/events/deployment/+/created	Containerized application	Create
Application update	\$hw/events/deployment/+/updated	Containerized application	Update
Node being brought online	\$hw/events/edgeNode/+/online	Edge node	Bring online
Node being brought offline	\$hw/events/edgeNode/+/offline	Edge node	Bring offline

## System Subscription Process

The system subscription process is as follows:

1. Create an API on API Gateway for IEF to call.
2. Create an API Gateway endpoint on IEF.
3. Create a system subscription on IEF.

## Creating an API

Create an API on API Gateway. For details, see [Getting Started](#).

This API is called by IEF to send system event messages.

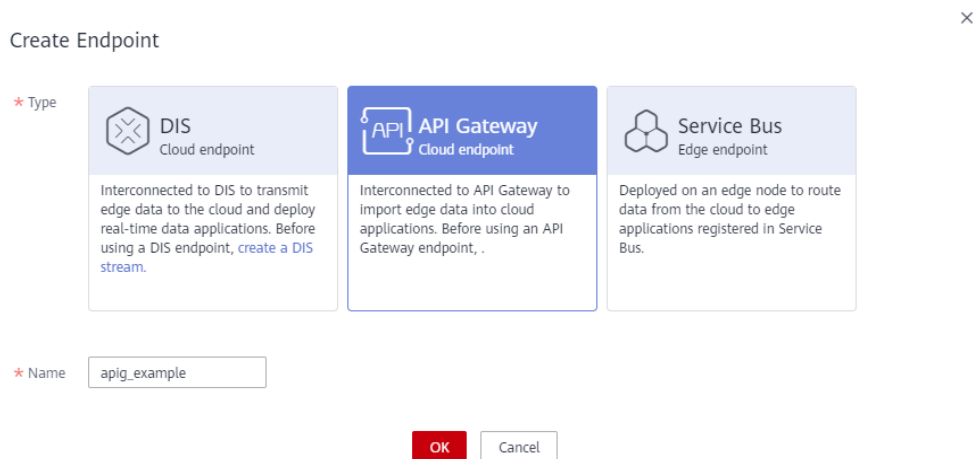
## Creating an API Gateway Endpoint

**Step 1** Log in to the IEF console.

**Step 2** In the navigation pane, choose **Edge-Cloud Messages > Endpoints**.

**Step 3** Click **Create Endpoint** in the upper right corner, and set the endpoint parameters.

**Figure 3-60** Creating an endpoint



- **Type:** Select **API Gateway**.
- **Name:** Enter an endpoint name.

**Step 4** Click **OK**. The endpoint is successfully created and the endpoint list page is displayed.

----End

## Creating a System Subscription

**Step 1** Log in to the IEF console.

**Step 2** In the navigation pane, choose **Edge-Cloud Messages > System Subscriptions**.

**Step 3** Click **Create System Subscription** in the upper right corner, and set the system subscription parameters.

**Figure 3-61** Creating a system subscription

\* Name

\* Topic   ⊕

\* Destination Endpoint

\* Destination Endpoint Resource   
The domain name in the URL must be configured in API Gateway. For details, see [Binding a Domain Name](#).

Description   
0/255

- **Name:** Enter a system subscription name.

**CAUTION**

System subscriptions and **message routes** are of the same resource type. Their names cannot conflict with each other.

- **Topic:** Select the resource and operation to be subscribed, for example, creating a containerized application. For details about the events that can be subscribed to, see [Table 3-12](#).
- **Destination Endpoint:** Select the endpoint created in [Creating an API Gateway Endpoint](#).
- **Destination Endpoint Resource:** Select the API created in [Creating an API](#).

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

----End

**Message Forwarding After Subscription**

After a system subscription is created, IEF will forward a message when a system event occurs. The statistics about message forwarding can be viewed on the **System Subscriptions** page of the IEF console.

**Figure 3-62** Number of forwarded messages

<input type="checkbox"/>	Name	Topic	Destination Endpoint	Status	Forwarded Messages
<input type="checkbox"/>	subscription	deployment/created	apig_example <span>Cloud</span> https://4e93a29712a245d2b700b70b0316ab0...	Enabled	Total: 0 Successful: 0 Failed: 0

Meanwhile, IEF will call the API registered on API Gateway. The request body is as follows: (The request body uses the standard CloudEvents format. For details, see [CloudEvents](#).)

```
{
  "data": {
    "event_type": "instance",
    "operation": "created",
    "timestamp": 134567677,
    "topic": "$hw/events/deployment+/created",
    "name": "xxx",
    "attributes": {"ID":"x"}
  },
  "datacontenttype": "application/json",
  "source": "sysevents",
  "id": "xxx",
  "time": "2020-11-5 xxx"
}
```

- **data:** system event data.
  - **event\_type:** resource type. The value is a character string.
  - **operation:** operation performed on the resource. The value is a character string.
  - **topic:** topic to which a message is sent. The value is a character string.
  - **timestamp:** time when an event occurs. The value is of the UInt64 type.
  - **name:** resource name. The value is a character string.
  - **attributes:** resource attributes. This parameter is not contained in the request for deleting a resource. The value is of the Object type.
- **datacontenttype:** format of the system event data content. The value is a character string.
- **source:** source of the system event. The value is a character string.
- **id:** system event ID, which is a character string.
- **time:** time when the system event occurs. The value is a character string.

[Table 3-12](#) lists the resource types, operations, and topics supported by IEF.

## 3.5 Batch Management

### 3.5.1 Registering Nodes in Batches

#### Scenario

If you need to manage, update, and maintain a large number of edge nodes of the same type, it will be labor-consuming to do so one by one as described in [Registering an Edge Node](#).

IEF provides batch node registration to pre-install required software on edge nodes of the same type so that they can be connected to the network after they are powered on. This one-to-many design improves management efficiency and reduces O&M costs.

## Procedure of Registering Nodes in Batches

The procedure for managing edge nodes in batches is as follows:

1. Prepare edge nodes that meet certain requirements. Then, configure the environment on the edge nodes by referring to [Configuring the Edge Node Environment](#).
2. After [creating a batch node registration job](#), obtain the configuration file and registration software, and pre-install them on the edge nodes. For details, see [Managing Edge Nodes in Batches](#). If the registration command is configured in the startup script, the device will be automatically managed as an edge node of IEF after being powered on and connected to the network.

## Creating a Batch Node Registration Job

**Step 1** Log in to the IEF console.

**Step 2** In the navigation pane, choose **Batch Management > Node Registration Jobs**, and click **Register Edge Nodes** in the upper right corner of the displayed page.

**Step 3** Configure basic information. The parameters marked with \* are mandatory.

The screenshot shows a form for creating a batch node registration job. At the top, it indicates the 'Service Instance' is 'Professional Service Instance'. The 'Name' field is mandatory (marked with an asterisk) and contains the value '1'. The 'Description' field is a large text area with a placeholder 'Enter a description.' and a character count of '0/255'. Below the description is a 'Tags' section with two input fields for 'Tag key' and 'Tag value', and a note 'You can add 20 more tags.' At the bottom, there is a 'Properties' section with a table structure for 'Key', 'Value', and 'Operation', and an 'Add Property' button.

- **Name:** name of the batch node registration job.
- **Description:** description of the job.
- **Tags:** Tags can be used to classify resources, facilitating resource management.
- **Properties:** A property is in key-value pair format. Enter a key and value.

**Step 4** Click **Register** in the lower right corner. You can choose to register the edge nodes using a certificate or a token.

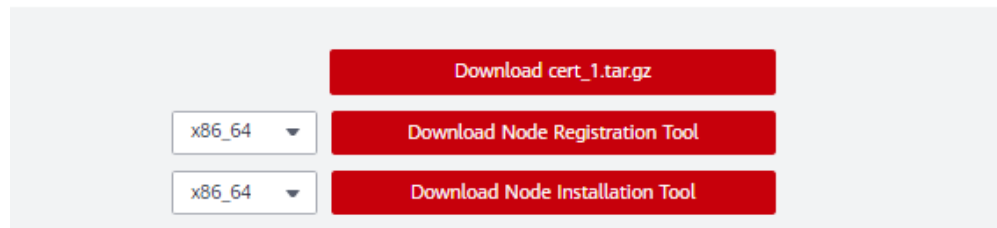
- **Method 1:** registration using a certificate: Download the configuration file, EdgeCore Register, and EdgeCore Installer.

**NOTICE**

To enhance node security, you must download the configuration file and tools now. The configuration file and tools cannot be retrieved later.

Download the product certificate and tools to complete the creation.

Download the certificate right now because you will not be able to retrieve it afterwards.

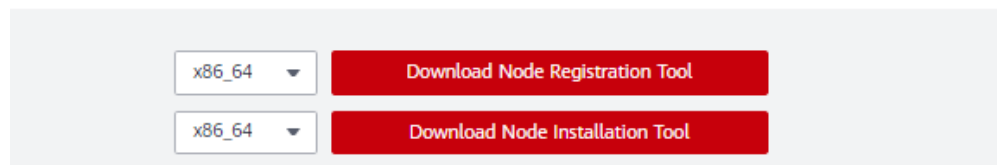


- a. Click **Download cert\_1.tar.gz** to download the configuration file.
  - b. Select the CPU architecture of your edge nodes, and click **Download Node Registration Tool** and **Download Node Installation Tool**.
- Method 2: registration using a token: Download EdgeCore Register and EdgeCore Installer and save the installation credential.

**Figure 3-63** Downloading EdgeCore Register and EdgeCore Installer

Download the product certificate and tools to complete the creation.

To connect a node to an IEF, perform the following steps.



**Figure 3-64** Installation credential

**4) Run the registration command.**

```
cd /opt/edge-register; ./register --mode=identifier --identifier=eyJ1cmwiOiJodHRwczovLzEw
MC44NS4xMTguMzI6NDQzIiwiaXV0aCI6IlFtVmhmaUJsZVVwb1lrZGphVTlwU2xOVmVra3hUb
WxKYzBsdVVqVmpRMGsyU1d0d1dGwkrTamt1WlhsS2JHVklrV2xQYWxFMFRxcE5lazU2VlR
CTUjGVnpTVzFzYTBscWlybFBSR2hxVFVSWk1sbDZaM1JOYWtwb1dsTXdNRmx0VFhsTVZHY3
dUMGRKZEZsNlFUQmFiVnBwVFZSQ2FWcHFwbnhKbXhKcFlWYzFlbVJlUm5WwWk1sWm1ZV
```

- a. Select the CPU architecture of your edge nodes, and click **Download Node Registration Tool** and **Download Node Installation Tool**.
- b. Save the **identifier** field of the installation command.

**Step 5** Upload the downloaded file to the edge nodes, and then manage the edge nodes on IEF by referring to [Managing Edge Nodes in Batches](#).

----End

## Managing Edge Nodes in Batches

Follow the procedure of the finish step on the **Register Edge Nodes** page to perform the batch management operation to the edge nodes.

**Step 1** Log in to an edge node as a user with sudo permissions.

**Step 2** Upload the configuration files and tools downloaded in step [Step 4](#) of "Creating a Batch Node Registration Job" to the edge nodes.

**Step 3** Run the following commands to decompress the configuration file, EdgeCore Register, and EdgeCore Installer:

```
sudo tar -zxvf edge-installer_1.0.10_x86_64.tar.gz -C /opt
```

```
sudo tar -zxvf edge-register_2.0.10_x86_64.tar.gz -C /opt
```

```
sudo tar -zxvf cert_batchNodes.tar.gz -C /opt/edge-register/ (This command is not required for registration using a token.)
```

Replace *edge-installer\_1.0.10\_x86\_64.tar.gz*, *edge-register\_2.0.10\_x86\_64.tar.gz*, and *cert\_batchNodes.tar.gz* with the names of the files downloaded in [Creating a Batch Node Registration Job](#).

**Step 4** (Optional) Customize the node information.

For batch node management, the default configuration will be applied for these edge nodes. Where:

- The node name is the combination of the host name and MAC address of the node, that is, **Host name\_MAC address**.
- The GPU and NPU options are disabled.
- The Docker function is enabled.

If you want to customize node information, configure the **nodeinfo** file. Run the following commands to open the **nodeinfo** file:

```
cd /opt/edge-register
```

```
vi nodeinfo
```

Specify the node information in the **nodeinfo** file. The following example lists all configurable parameters. You can set the parameters based on site requirements and delete the parameters that do not need to be customized.

Parameters such as the node name, descriptions, GPU enabled or not, NPU, NPU types, attributes, and log configuration are included. For details about the parameters, see [API Reference > Registering an Edge Node](#).

```
{  
  "name": "nodename",  
  "description": "",  
  "enable_gpu": false,  
  "enable_npu": true,  
  "npu_type": "****",  
  "enable_docker": true,  
}
```



```
"attributes": [
  {
    "key": "key1",
    "value": "value1"
  }
],
"log_configs": [
  {
    "component": "app",
    "type": "local",
    "level": "debug",
    "size": 100,
    "rotate_num": 5,
    "rotate_period": "daily"
  }
],
"device_infos": [
  {
    "device_ids": ["15696983-5ee6-43b4-9653-5d8512813dcc"],
    "relation": "camera",
    "comment": "devicedescription"
  }
],
"mqtt_config": {
  "enable_mqtt": false,
  "mqttps": []
},
"tags": [
  {
    "key": "name",
    "value": "value"
  }
]
}
```

**Step 5** Run one of the following commands to manage the edge nodes:

- For edge nodes registered using certificates:  
**cd /opt/edge-register; ./register --mode=cert**
- For edge nodes registered using tokens:  
**cd /opt/edge-register; ./register --mode=identifier --identifier=Credential**  
*for registration using a token*  
Replace *Credential for registration using a token* with the value of **identifier** field of the installation credential saved in [Creating a Batch Node Registration Job](#).

After the command is executed, check whether the node is successfully registered by referring to [Viewing Managed Nodes](#).

You can also add the registration command **cd /opt/edge-register; ./register --mode=cert** to the startup script. In this way, the edge node can automatically run the registration command when it is powered on.

If the command output is **0**, the edge node is successfully registered on IEF.

----End

## Viewing Managed Nodes

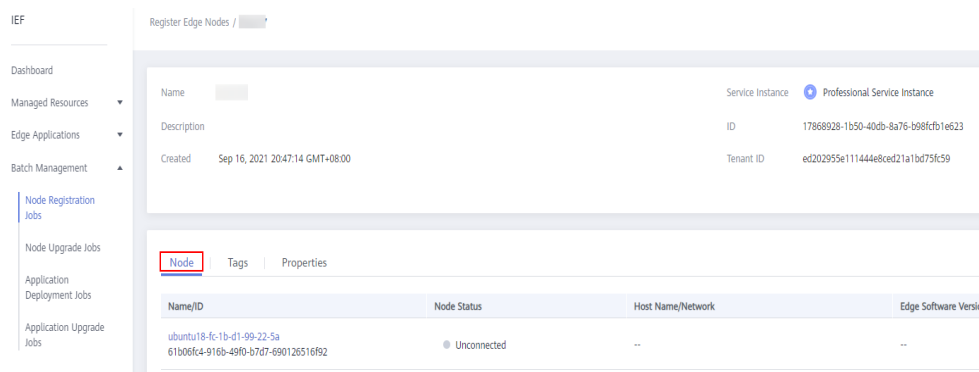
You can view the nodes managed in batches on the console.

**Step 1** Log in to the IEF console.

**Step 2** In the navigation pane, choose **Batch Management > Node Registration Jobs**, and click a job name on the displayed page.

**Step 3** You can view the managed nodes on the **Nodes** tab.

**Figure 3-65** Viewing nodes managed in batches



----End

## 3.5.2 Upgrading Nodes in Batches

### Scenario

In some scenarios, you may need to manage, update, and maintain EdgeCore software of a large number of edge nodes. IEF allows upgrading edge nodes in batches for this purpose.

#### NOTE

Edge nodes can be upgraded successfully only when they communicate with IEF properly.

### Precautions

- IEF does not proactively upgrade EdgeCore on your edge nodes. You are advised to upgrade the node manually in the time window with the minimum impact on your services.
- Upgrading a version within the maintenance period will not interrupt your applications running on the edge node. However, if message routing is used, services may be temporarily affected.
- Upgrading a version beyond the maintenance period may temporarily interrupt services due to container restart.
- Do not change node configurations during the node upgrade, such as restarting Docker, installing or uninstalling the GPU/NPU driver, upgrading the OS kernel, or modifying network configurations, which may result in node upgrade failures.

### Procedure

**Step 1** Log in to the IEF console.

**Step 2** In the navigation pane, choose **Batch Management > Node Upgrade Jobs**, and click **Create Node Upgrade Job** in the upper right corner of the displayed page.

**Step 3** Configure basic information about the batch node upgrade job.

**Figure 3-66** Upgrading nodes in batches

\* Name

Description   
0/255

Tags    
You can add 20 more tags.

---

### Upgrade Configuration

Upgrade Object

- **Name:** name of the node upgrade job to be created.
- **Description:** description of the node upgrade job.
- **Tags:** tag of the node upgrade job.
- **Upgrade Object:** node to be upgraded.

Click **Select Edge Node**, and select the nodes to be upgraded.

You can also click **Search by Tag** in the upper right corner of the page, enter the tag key and value, and click **Search** to filter out the edge nodes with the specified tag. Then, select the edge nodes to be upgraded and click **OK**.

**Step 4** Click **Next**. Confirm the upgrade information and click **Create**.

----End

## Status Description

A batch node upgrade job can be in any of the following states.

- **Pending:** The job is waiting to be executed.
- **Running:** The job is being executed.

- **Successful:** All tasks in the job are executed successfully.
- **Partially successful:** Some tasks in the job are executed successfully.
- **Failed:** All tasks in the job failed.
- **Stopping:** The job is being stopped.
- **Stopped:** The job is stopped.
- **Update timed out:** The job is pended for more than 10 minutes or the job has not completed even after 10 minutes.

A job can be stopped during execution and resumed after being stopped.

If a job fails to be executed, partially succeeds, or times out, you can retry the job.

**Figure 3-67** Retry

Name/ID	Status	Results	Created	Updated	Description	Operation
idf-6d5a3857-50df-4c2d-a7a6-7631cf52071d	Success	Total 1 Successful 1 Failed 0 Pending 0	Sep 30, 2022 18:24:34 GMT+08:00	Sep 30, 2022 18:24:54 GMT+08:00	adaadad	Stop   Resume   More
awdowdvwad-0fc6b8c2-1369-48aa-809e-ed4a3a8ca538	Partially su...	Total 2 Successful 1 Failed 1 Pending 0	Sep 26, 2022 20:46:28 GMT+08:00	Sep 26, 2022 20:46:35 GMT+08:00	adaw	Stop   Resume   More
awdowd-0ac1079f-9d3d-4d46-9d37-9d713d6d761	Success	Total 1 Successful 1 Failed 0 Pending 0	Sep 26, 2022 20:39:23 GMT+08:00	Sep 26, 2022 20:39:28 GMT+08:00	awd	Stop   Resume   Delete

## 3.5.3 Deploying Applications in Batches

### Scenario

In some scenarios, you may need to deploy the same applications on a large number of edge nodes. IEF allows deploying applications in batches for this purpose.

#### NOTE

- All edge nodes must use the same architecture, for example, x86 or Arm.
- The architecture of the container image to be used must be the same as that of the edge node on which a containerized application is to be deployed. For example, if the edge node uses x86, the container image must also use x86.

### Procedure

- Step 1** Log in to the IEF console.
- Step 2** In the navigation pane, choose **Batch Management** > **Application Deployment Jobs** and click **Create Application Deployment Job** in the upper right corner of the displayed page.
- Step 3** Specify basic information about the application deployment job.

**Figure 3-68** Creating a batch application deployment job

The screenshot shows a multi-step wizard for creating a batch application deployment job. The current step is 'Specify Basic Information'. The form contains the following elements:

- Service Instance:** Professional Service Instance
- Name:** A text input field with the placeholder 'Enter a job name.'
- Description:** A text area with the placeholder 'Enter a job description.' and a character count of 0/255.
- Tags:** Two input fields for 'Tag key' and 'Tag value', with a note 'You can add 20 more tags.'
- Deployment Configuration:**
  - Deployment Object:** 'Manually specify' is selected, with a 'Select Edge Node' button below it.
  - Advanced Settings:**
    - Restart Policy:** 'Always restart' is selected. A tooltip states: 'The system restarts the container regardless of whether it had quit normally or abnormally.'
    - Host PID:** 'Disable' is selected.

At the bottom right, there are 'Cancel' and 'Next' buttons.

- **Name:** name of the application deployment job to be created.
- **Description:** description of the application deployment job.
- **Tags:** tag of the application deployment job.
- **Deployment Object:** edge node where the containerized application is to be deployed.

Click **Select Edge Node**, and select target edge nodes.

You can also click **Search by Tag** in the upper right corner of the page, enter the tag key and value, and click **Search** to filter out the edge nodes with the specified tag. Then, select edge nodes and click **OK**.

**Figure 3-69** Selecting tags

The screenshot shows the 'Select Edge Node' dialog box. It includes a search bar with 'Tag key' and 'Value' fields, a '+' button, and a search button. Below the search bar, there is a list of tags, with 'os=ubuntu' selected. At the bottom, there are 'Search' and 'Reset' buttons.

<input type="checkbox"/>	Name/ID	Node Status	Host Name/Network	Created
<input type="checkbox"/>	ecs-lef-ubuntu_fa-16-3e-28-07-a0 209ba62d-2588-4476-ae1-6617e93bad14	Running	ecs-lef-ubuntu 192.168.0.204	Aug 05, 2021 14:40:17 GMT+08:00

At the bottom of the dialog, there are 'OK' and 'Cancel' buttons.

- **Restart Policy**
  - **Always restart:** The system restarts the container regardless of whether it had quit normally or unexpectedly.
  - **Restart upon failure:** The system restarts the container only if it had previously quit unexpectedly.
  - **Do not restart:** The system does not restart the container regardless of whether it had quit normally or unexpectedly.

 **NOTE**

Only containerized applications with **Always restart** selected can be upgraded after being created.

- **Host PID:** Enabling this function allows containers to use the host's PID namespace. This function can be enabled only on edge nodes whose software version is 2.8.0 or later.

**Step 4** Click **Next**. Then, configure application deployment information.

**Figure 3-70** Application deployment information

The screenshot shows a configuration form for application deployment. The fields are as follows:

- Service Instance:** Professional Service Instance (selected with a star icon)
- Creation Mode:** Custom
- Name Prefix:** batch-deploy
- Instances:** 1 (with minus and plus buttons)
- Configuration Method:** Custom (selected), Application template
- Description:** Enter a description. (0/255 characters)
- Tags:**  Inhibit common tags from edge nodes. Below are input fields for Tag key and Tag value. A note says "You can add 20 more tags."

- **Name Prefix:** prefix of the application deployment name. IEF generates a complete name based on the prefix.
- **Instances:** number of instances deployed for the application. This parameter can only be set to **1** for a professional service instance.
- **Configuration Method**
  - **Custom:** Configure the application step by step. For details, see [Step 5](#).
  - **Application template:** Select a predefined application template and modify it based on your requirements. This method reduces repeated operations. The configurations in a template are the same as those set in [Step 5](#). For details on how to create a template, see [Application Templates](#).
- **Description:** description of the containerized application.

- **Tags:** tag of the containerized application. You can select **Inhibit common tags from edge nodes**.

**Step 5** Configure containers.

Click **Use Image** under the image to be used to deploy the application.

- **My Images:** Displays all the images you have created in **SWR**.
- **Shared Images:** Displays images shared by other users. Shared images are managed and maintained in SWR. For details, see **Sharing a Private Image**.

After selecting an image, specify container configurations.

- **Image Version:** Select the version of the image to be used to deploy the application.

**NOTICE**

Do not use version **latest** when deploying containers in the production environment. Because this will make it difficult to determine the version of the running image and to roll back the application properly.

- **Container Specifications:** Specify CPU, memory, Ascend AI accelerator card, and GPU quotas.
- **Ascend AI accelerator card:** The AI accelerator card configuration of the containerized application must be the same as that of the edge node actually deployed. Otherwise, the application will fail to be created. For details, see **how to configure the AI accelerator card during edge node registration**.

 **NOTE**

For NPUs after virtualization partition, only one virtualized NPU can be mounted to a container. The virtualized NPU can be allocated to another container only after the original container quits.

The following table lists the NPU types supported by Ascend AI accelerator cards.

**Table 3-13** NPU types

Type	Description
Ascend 310	Ascend 310 chips
Ascend 310B	Ascend 310B chips

**Figure 3-71** Container configuration

The screenshot displays the container configuration interface. At the top, the 'Image Name' is set to 'wordpress' with a 'Change Image' link. Below it, the 'Image Version' is set to 'php7.3' and the 'Container Name' is 'container-14944d3c'. The 'Container Specifications' section is divided into two parts: CPU and Memory. For CPU, both 'Request' and 'Limit' are set to '0.25 cores'. For Memory, both 'Request' and 'Limit' are set to '512.00 MIB'. At the bottom, the 'AI Accelerator Card' section shows 'Not Installed' in a blue box, with 'Ascend AI accelerator card' and 'NVIDIA GPU' as alternative options.

You can also configure the following advanced settings for the container:

- **Command**

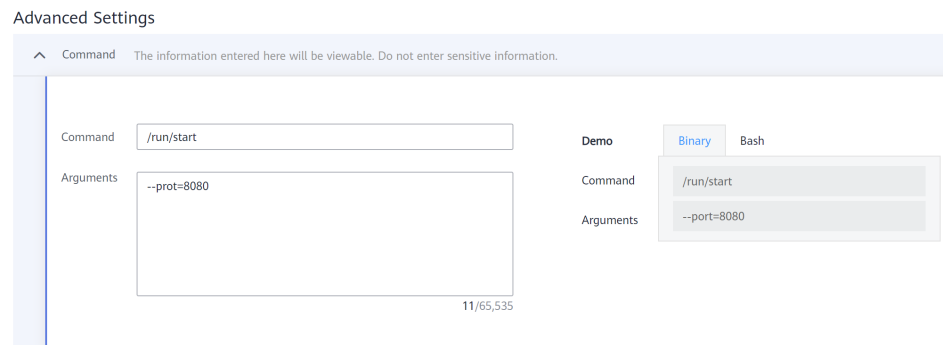
A container image has metadata that stores image details. If lifecycle commands and arguments are not set, IEF runs the default commands and arguments provided during image creation, that is, **ENTRYPOINT** and **CMD** in the Dockerfile.

If the commands and arguments used to run a container are set during application creation, the default commands **ENTRYPOINT** and **CMD** are overwritten during image building. The rules are as follows:

**Table 3-14** Commands and arguments used to run a container

Image ENTRYPOINT	Image CMD	Command to Run a Container	Arguments to Run a Container	Command Executed
[touch]	[/root/test]	Not set	Not set	[touch /root/test]
[touch]	[/root/test]	[mkdir]	Not set	[mkdir]
[touch]	[/root/test]	Not set	[/opt/test]	[touch /opt/test]
[touch]	[/root/test]	[mkdir]	[/opt/test]	[mkdir /opt/test]



**Figure 3-72** Command

### – Command

Enter an executable command, for example, **/run/start**.

If there are multiple commands, separate them with spaces. If the command contains a space, enclose the command in quotation marks ("").

#### NOTE

In the case of multiple commands, you are advised to run **/bin/sh** or other **shell** commands. Other commands are used as arguments.

### – Arguments

Enter the argument that controls the container running command, for example, **--port=8080**.

If there are multiple arguments, separate them with line breaks.

## • Security Options

- You can enable **Privileged Mode** to grant root permissions to the container for accessing host devices (such as GPUs and FPGAs).

### – RunAsUser Switch

By default, IEF runs the container as the user defined during image building.

You can specify a user to run the container by turning this switch on, and entering the user ID (an integer ranging from 0 to 65534) in the text box displayed. If the OS of the image does not contain the specified user ID, the application fails to be started.

## • Environment Variables

An environment variable affects the way a running container will behave. Variables can be modified after workload deployment. Currently, environment variables can be manually added, imported from secrets or ConfigMaps, or referenced from **hostIP**.

- **Added manually:** Customize a variable name and value.
- **Added from Secret:** You can customize a variable name. The variable value is referenced from secret configuration data. For details on how to create a secret, see [Secrets](#).
- **Added from ConfigMap:** You can customize a variable name. The variable value is referenced from ConfigMap configuration data. For details on how to create a ConfigMap, see [ConfigMaps](#).

- **Variable reference:** The variable value is referenced from **hostIP**, that is, the IP address of an edge node.

 **NOTE**

IEF does not encrypt the environment variables you entered. If the environment variables you attempt to configure contain sensitive information, you need to encrypt them before entering them and also need to decrypt them when using them.

IEF does not provide any encryption and decryption tools. If you need to configure cypher text, choose your own encryption and decryption tools.

- **Data Storage**

You can define a local volume and mount the local storage directory of the edge node to the container for persistent data storage.

Currently, the following four types of local volumes are supported:

- **hostPath:** used for mounting a host directory to the container. **hostPath** is a persistent volume. After an application is deleted, the data in **hostPath** still exists in the local disk directory of the edge node. If the application is re-created later, previously written data can still be read after the directory is mounted.

You can mount the application log directory to the **var/IEF/app/log/{appName}** directory of the host. In the directory name, **{appName}** indicates the application name. The edge node will upload the .log and .trace files in the **/var/IEF/app/log/{appName}** directory to AOM. The mount directory is the log path of the application in the container. For example, the default log path of the Nginx application is **/var/log/nginx**. The permission must be set to **Read/Write**.

**Figure 3-73** Log volume mounting

Local Volume Name	Type	Mount Directory	Permission	Operation
log	hostPath	/var/IEF/app/log/ngi	/var/log/nginx	Read/Write Delete

- **emptyDir:** a simple empty directory used for storing transient data. It can be created in hard disks or memory. **emptyDir** is an empty directory after being mounted. The application can read files from and write files into the directory. **emptyDir** has the same lifecycle as the application. If the application is deleted, the data in **emptyDir** is deleted along with it.
- **configMap:** a type of resources that store configuration details required by the application. For details on how to create a **ConfigMap**, see [ConfigMaps](#).
- **secret:** a type of resources that store sensitive data, such as authentication, certificate, and key details. For details on how to create a secret, see [Secrets](#).

**NOTICE**

- The container path cannot be a system directory, such as / or /var/run. Otherwise, an exception occurs. You are advised to mount the container to an empty directory. If the directory is not empty, ensure that the directory does not contain any files that affect container startup. Otherwise, the files will be replaced, making it impossible for the container to be properly started. As a result, the application creation will fail.
- If the container is mounted into a high-risk directory, you are advised to use an account with minimum permissions to start the container. Otherwise, high-risk files on the host machine may be damaged.

**Health Check**

Health check regularly checks the status of containers or workloads.

- **Liveness Probe:** The system executes the probe to check if a container is still alive, and restarts the instance if the probe fails. Currently, the system probes a container by HTTP request or command and determines whether the container is alive based on the response from the container.
- **Readiness Probe:** The system invokes the probe to determine whether the instance is ready. If the instance is not ready, the system does not forward requests to the instance.

For details, see [Health Check Configuration](#).

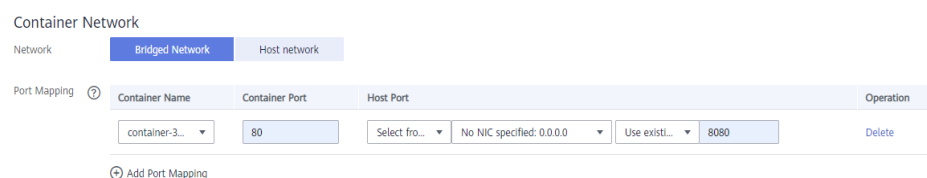
**Step 6** Click **Next**.

Containers can be accessed through a bridged network or a host network.

**Bridged network**

The container uses an independent virtual network. A mapping between container and host ports needs to be configured to enable the external communication. After port mapping is configured, traffic destined for a host port is distributed to the mapping container port. For example, if container port 80 is mapped to host port 8080, the traffic destined for host port 8080 will be directed to container port 80.

You can select a host NIC to enable the port bound to this NIC to communicate with the container port. Note that port mapping does not support NICs with IPv6 addresses.

**Figure 3-74** Bridged network**Host network**

The network of the host (edge node) is used. To be specific, the container and the host use the same IP address, and network isolation is not required between them.

**Step 7** Click **Next**. Confirm the specifications of the containerized application and click **Create**.

----End

## Status Description

A batch application deployment job can be in any of the following states.

- **Pending:** The job is waiting to be executed.
- **Running:** The job is being executed.
- **Successful:** All tasks in the job are executed successfully.
- **Partially successful:** Some tasks in the job are executed successfully.
- **Failed:** All tasks in the job failed.
- **Stopping:** The job is being stopped.
- **Stopped:** The job is stopped.
- **Update timed out:** The job is pended for more than 10 minutes or the job has not completed even after 10 minutes.

A job can be stopped during execution and resumed after being stopped.

If a job fails to be executed, partially succeeds, or times out, you can retry the job.

## 3.5.4 Upgrading Applications in Batches

### NOTE

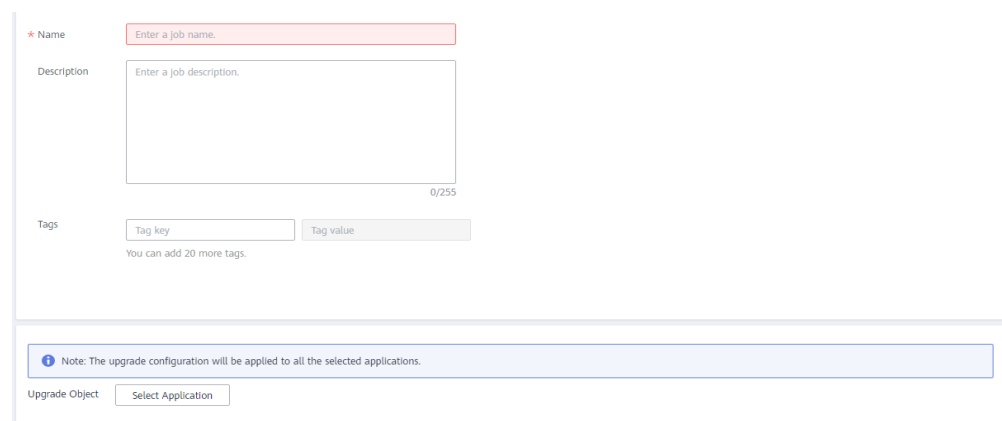
The architecture of the container image to be used must be the same as that of the edge node on which a containerized application is to be deployed. For example, if the edge node uses x86, the container image must also use x86.

**Step 1** Log in to the IEF console.

**Step 2** In the navigation pane, choose **Batch Management > Application Upgrade Jobs**, and click **Create Application Upgrade Job** in the upper right corner of the displayed page.

**Step 3** Specify basic information about the application upgrade job.

**Figure 3-75** Upgrading applications in batches



The screenshot shows a web form for creating an application upgrade job. It contains the following elements:

- Name:** A text input field with a red border and placeholder text "Enter a job name."
- Description:** A larger text area with a placeholder "Enter a job description." and a character count "0/255" at the bottom right.
- Tags:** A section with two input fields labeled "Tag key" and "Tag value", and a note "You can add 20 more tags."
- Note:** A blue-bordered box containing the text "Note: The upgrade configuration will be applied to all the selected applications."
- Upgrade Object:** A button labeled "Select Application" located below the note.

- **Name:** name of the application upgrade job to be created.
- **Description:** description of the application upgrade job.
- **Tags:** tag of the application upgrade job.
- **Upgrade Object:** containerized application to be upgraded.

Click **Select Application**, and select applications to be upgraded.

You can also click **Search by Tag** in the upper right corner of the page, enter the tag key and value, and click **Search** to filter out the containerized applications with the specified tag. Then, select the containerized applications to be upgraded and click **OK**.

**Step 4** Click **Next** and configure container information. The configurations are the same as that in **Step 5**.

**Step 5** Click **Next** and configure access settings. The configurations are the same as that in **Step 6**.

**Step 6** Click **Next**. Confirm the specifications of the containerized application and click **Create**.

----End

## Status Description

A batch application upgrade job can be in any of the following states.

- **Pending:** The job is waiting to be executed.
- **Running:** The job is being executed.
- **Successful:** All tasks in the job are executed successfully.
- **Partially successful:** Some tasks in the job are executed successfully.
- **Failed:** All tasks in the job failed.
- **Stopping:** The job is being stopped.
- **Stopped:** The job is stopped.
- **Update timed out:** The job is pended for more than 10 minutes or the job has not completed even after 10 minutes.

A job can be stopped during execution and resumed after being stopped.

If a job fails to be executed, partially succeeds, or times out, you can retry the job.

## 3.6 Auditing

### 3.6.1 IEF Operations Supported by CTS

#### Scenario

Cloud Trace Service (CTS) is a log audit service provided by Huawei Cloud and intended for cloud security. It allows you to collect, store, and query cloud resource operation records and use these records for security analysis, compliance auditing, resource tracking, and fault locating.

With CTS, you can record operations associated with IEF for future query, audit, and backtrack operations.

## Key Operations Recorded by CTS

**Table 3-15** IEF operations that can be recorded by CTS

Operation	Resource Type	Trace Name
Registering an edge node	node	createEdgeNode
Updating an edge node	node	updateEdgeNode
Deleting an edge node	node	deleteEdgeNode
Creating a device template	deviceTemplate	createDeviceTemplate
Updating a device template	deviceTemplate	updateDeviceTemplate
Deleting a device template	deviceTemplate	deleteDeviceTemplate
Registering a device	device	createDevice
Updating a device	device	updateDevice
Deleting a device	device	deleteDevice
Deploying an Edge-Connector application	device	deployConnector
Updating device twins	device	updateDeviceTwin
Updating the device access configuration	device	updateDeviceAccessConfig
Creating an application template	application	createApplication
Updating an application template	application	updateApplication
Deleting an application template	application	deleteApplication
Creating an application template version	appVersion	createAppVersion
Updating an application template version	appVersion	updateAppVersion
Deleting an application template version	appVersion	deleteAppVersion
Creating a ConfigMap	configmap	createConfigMap

Operation	Resource Type	Trace Name
Updating a ConfigMap	configmap	updateConfigMap
Deleting a ConfigMap	configmap	deleteConfigMap
Creating a containerized application	deployment	createDeployment
Updating a containerized application	deployment	updateDeployment
Deleting a containerized application	deployment	deleteDeployment
Querying a list of containerized applications	deployment	getDeploymentList
Querying a containerized application	deployment	getDeployment
Creating an endpoint	endpoint	createEndpoint
Deleting an endpoint	endpoint	deleteEndpoint
Adding a node certificate	node	AddNodeCert
Deleting a node certificate	node	deleteNodeCert
Upgrading the firmware	nodeFirmware	UpgradeNodeFirmware
Querying a list of application instances	Pods	getPods
Querying an instance	Pod	getPod
Creating a node registration job	product	createProduct
Deleting a node registration job	product	deleteProduct
Creating a node upgrade job	batchJob	createJob
Pausing a node upgrade job	batchJob	pauseJob
Deleting a node upgrade job	batchJob	deleteJob
Querying quotas	quota	getQuota
Updating a quota	quota	updateQuota
Creating a rule	rule	createRule

Operation	Resource Type	Trace Name
Deleting a rule	rule	deleteRule
Updating a rule	rule	updateRule
Creating a secret	secret	createSecret
Updating a secret	secret	updateSecret
Deleting a secret	secret	deleteSecret
Filtering resources by tag	Tags	filterTags
Adding or deleting tags in batches	Tags	batchAddDeleteTags
Adding a tag	Tags	addTag
Deleting a tag	Tags	deleteTag
Creating a system subscription	Systemevent	createSystemevent
Deleting a system subscription	Systemevent	DeleteSystemevent
Enabling a system subscription	Systemevent	startSystemevent
Disabling a system subscription	Systemevent	stopSystemevent

## 3.6.2 Viewing Tracing Logs

### Scenario

After you enable CTS, the system starts recording operations performed on IEF resources. CTS stores operation records generated within a week.

This section describes how to view the records on the CTS console.

### Procedure

- Step 1** Log in to the CTS console.
- Step 2** In the navigation pane, choose **Trace List**.
- Step 3** Set the filter criteria and click **Query**.

The following filters are available:


- **Trace Source, Resource Type, and Search By**

Select the desired filter criteria from the drop-down lists. Select **Management** for **Trace Type** and **IEF** for **Trace Source**.

In the preceding information:



- If you select **Resource ID** for **Search By**, you need to enter a resource ID. Only whole word match is supported.
- If you select **Resource name** for **Search By**, you need to select or enter a specific resource name.
- **Operator**: Select a specific operator from the drop-down list.
- **Trace Status**: Select one of **All trace statuses**, **Normal**, **Warning**, and **Incident**.
- Time range: You can select **Last 1 hour**, **Last 1 day**, **Last 1 week**, or **Customize**.

**Step 4** Click  on the left of a trace to expand its details.

**Step 5** Click **View Trace** in the upper right corner of the trace details area.

----End

## 3.7 Permissions Management

### 3.7.1 Creating a User and Granting Permissions

This section describes how to use **IAM** to implement fine-grained permissions control for your IEF resources. With IAM, you can:

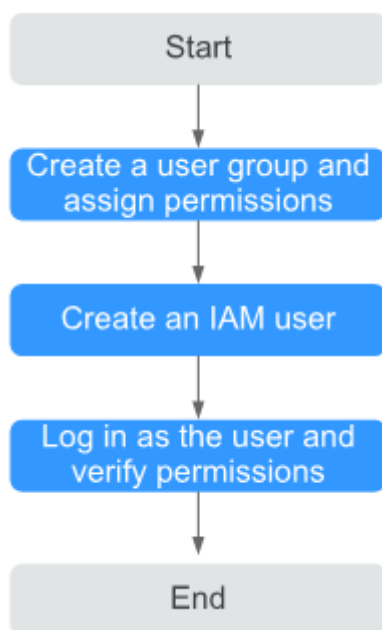
Entrust a cloud account or cloud service to perform efficient O&M on your IEF resources.

If your account does not need individual IAM users, you may skip over this section.

This section describes the procedure for granting permissions (see [Figure 3-76](#)).

#### Process Flow

**Figure 3-76** Process for granting IEF permissions



### 1. Create a user group and assign permissions.

Create a user group on the IAM console, and assign the **IEF ReadOnlyAccess** policy to the group. When assigning permissions to a user group, set **Scope** to **Region-specific projects**, and set parameters according to the following rules:

- To assign permissions in certain regions, select one or more specified projects, for example, **cn-north-4 [CN North-Beijing4]**. Note: If you select **All Projects** in this scenario, the authorization will not take effect.
- To assign permissions in all regions, select **All projects**.

**Figure 3-77** Assigning permissions in certain regions



**Figure 3-78** Assigning permissions in all regions



### 2. Create an IAM user and add the user to the user group.

Create a user on the IAM console and add the user to the group created in **1**.

### 3. Log in and verify permissions.

Log in to the IEF console by using the user created in **2**, and verify that the user has the administrator permissions for IEF.

## 3.7.2 IEF Custom Policies

IEF allows you to create custom policies.

You can create custom policies in either of the following ways:

- Visual editor: Select cloud services, actions, resources, and request conditions. This does not require knowledge of policy syntax.
- JSON: Edit JSON policies from scratch or based on an existing policy.

For details, see [Creating a Custom Policy](#). The following section contains examples of common IEF custom policies.

### Example Custom Policies

Allowing users to create and update applications and application templates

```
{
  "Version": "1.1",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ief:deployment:create",
```

```

        "ief:appVersion:update",
        "ief:deployment:update",
        "ief:application:create"
    ],
    "Condition": {
        "StringEquals": {
            "ief:AssumeUserName": [
                "test"
            ]
        }
    },
    "Resource": [
        "ief:*.deployment:*",
        "ief:*.appVersion:*",
        "ief:*.application:*"
    ]
}
]
}

```

### 3.7.3 IEF Resources

A resource is an object that exists within a service. IEF resources include product, node, group, deployment, batchjob, application, appVersion, and IEFInstance. You can select these resources by specifying their paths.

**Table 3-16** IEF resources and their paths

Resource	Resource Name	Path
product	Node registration job	IEF:*.product:
node	Edge node	IEF:*.node:
group	Edge node group	IEF:*.group:
deployment	Deployment	IEF:*.deployment:
batchjob	Job	IEF:*.batchjob:
application	Application template	IEF:*.application:
appVersion	Application template version	IEF:*.appVersion:
IEFInstance	IEF instance	IEF:*.IEFInstance:

### 3.7.4 IEF Request Conditions

Request conditions are useful in determining when a custom policy takes effect. A request condition consists of a condition key and operator. Condition keys are either global or service-level and are used in the Condition element of a policy

statement. **Global condition keys** (starting with **g:**) are available for operations of all services, while service-level condition keys (starting with a service name such as **ief:**) are available only for operations of a specific service. An operator is used together with a condition key to form a complete condition statement.

IEF has a group of predefined condition keys that can be used in IAM. For example, to define an "Allow" permission, you can use the condition key *ief:AssumeUserName* to filter matching requesters by username. The following table lists the predefined condition keys of IEF.

**Table 3-17** Predefined condition keys of IEF

Condition Key	Operator	Description
ief:AssumeUserName	StringEndWithAnyOfIfExists StringStartWithAnyOfIfExists StringEndWithIfExists StringStartWithIfExists StringNotLikeAnyOfIfExists StringLikeAnyOfIfExists StringNotEqualsIgnoreCaseAnyOfIfExists StringEqualsIgnoreCaseAnyOfIfExists StringNotEqualsAnyOfIfExists StringEqualsAnyOfIfExists StringNotLikeIfExists StringLikeIfExists StringNotEqualsIgnoreCaseIfExists StringEqualsIgnoreCaseIfExists StringNotEqualsIfExists StringEqualsIfExists IsNullOrEmpty StringEndWithAnyOf StringStartWithAnyOf StringEndWith StringStartWith StringNotLikeAnyOf StringLikeAnyOf StringNotEqualsIgnoreCaseAnyOf StringEqualsIgnoreCaseAnyOf	Used for matching username

Condition Key	Operator	Description
	StringNotEqualsAnyOf StringEqualsAnyOf StringNotLike StringLike StringNotEqualsIgnoreCase StringEqualsIgnoreCase StringNotEquals StringEquals	

## Example

This policy can be used only when the username is **test**.

```
{
  "Version": "1.1",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ief:deployment:create",
        "ief:appVersion:update",
        "ief:deployment:update",
        "ief:application:create"
      ],
      "Condition": {
        "StringEquals": {
          "ief:AssumeUserName": [
            "test"
          ]
        }
      }
    },
    {
      "Resource": [
        "ief:*:deployment:*",
        "ief:*:appVersion:*",
        "ief:*:application:*"
      ]
    }
  ]
}
```

### 3.7.5 Entrustment Description

IEF works closely with multiple cloud services to use image, storage, data, and monitoring functions. When you log in to the IEF console for the first time, IEF automatically requests permissions to access those cloud services in the region where you run your applications. Specifically:

- Application Operations Management (AOM)  
IEF can collect performance metrics and logs (optional) of edge nodes and application containers through AOM, helping you monitor the performance of edge nodes and applications in real time and quickly identify risks.

- Software Repository for Container (SWR)  
IEF can manage and download self-defined container images through SWR, helping you deploy containerized applications on edge nodes.
- Object Storage Service (OBS)  
IEF can access created edge functions through OBS, helping you deploy and run functions on edge nodes and devices.
- Data Ingestion Service (DIS)  
IEF can send data of edge nodes and devices to your DIS streams.

After you agree to the entrustment, IEF automatically creates an agency in IAM to delegate other resource operation permissions in your account to Huawei Cloud IEF. For details, see [Account Delegation](#).

The agencies automatically created in IAM are:

- [ief\\_admin\\_trust](#)
- [ief\\_edge\\_trust](#)

## ief\_admin\_trust

The `ief_admin_trust` agency has the Tenant Administrator permissions. Tenant Administrator has the administrator permissions on all cloud services except IAM. The permissions are used to call the cloud services that IEF depends on. The delegation takes effect only in the current region.

To use IEF in multiple regions, request for cloud resource permissions in each region. You can go to the IAM console, choose **Agencies**, and click **ief\_admin\_trust** to view the authorization records in each region.

### NOTE

IEF may fail to run as expected if the Tenant Administrator permissions are not assigned. So, do not delete or modify the **ief\_admin\_trust** agency when using IEF.

## ief\_edge\_trust

The `ief_edge_trust` agency does not contain the Tenant Administrator system role. It only has the operation permissions of cloud service resources required by IEF and is used by edge nodes to report monitoring, alarms, and logs.

If the permissions of the `ief_edge_trust` agency are different from those expected by IEF, the console displays a message indicating that the permissions have changed and a re-authorization is required.

You may need to modify the permissions of the `ief_edge_trust` agency in the following scenarios:

- The permissions on which IEF components depend may change with versions. For example, if a new component depends on new permissions, IEF will update the expected permission list and you need to grant the required permissions to `ief_edge_trust`.
- When you manually modify the permissions of the `ief_edge_trust` agency, the permissions of the agency are different from those expected by IEF. In this case, a message is displayed, asking you to re-authorize the agency. If a re-authorization, the permissions you previously modified may become invalid.





# 4 User Guide (Platinum)

---

## 4.1 Node Management

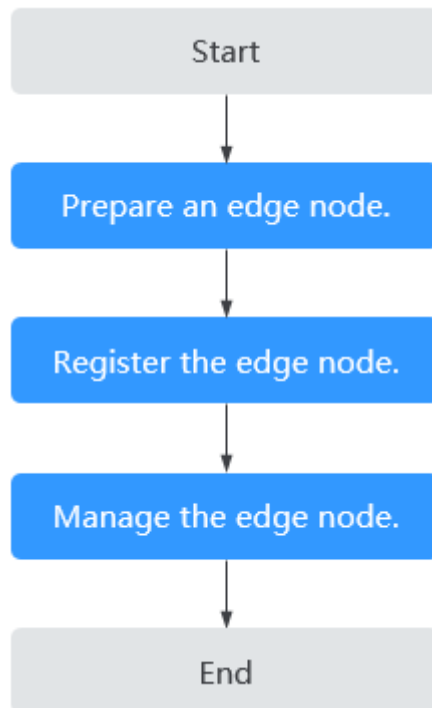
### 4.1.1 Edge Node Overview

Edge nodes are your own edge computing devices. They are used to run edge applications, process your data, and collaborate with cloud applications securely and conveniently. You can deploy system applications using IEF to extend cloud service capabilities to edge nodes, or deploy your own applications to implement edge computing capabilities.

To enable IEF to manage your edge node, perform the following steps:

1. Prepare the node. The edge node must meet requirements listed in [Configuring the Edge Node Environment](#).
2. Register the node on IEF, and download the EdgeCore installer and configuration file. For details, see [Registering an Edge Node](#).
3. Manage the edge node using the EdgeCore installer and configuration file. For details, see [Managing an Edge Node](#).

**Figure 4-1** Procedure



## 4.1.2 Configuring the Edge Node Environment

### Specifications Requirements

An edge node can be a physical machine or a virtual machine (VM). Edge nodes must meet the specifications listed in [Table 4-1](#).

**Table 4-1** Edge node requirements

Item	Specifications
OS	<p>The language of the operating system must be <b>English</b>.</p> <ul style="list-style-type: none"> <li>• x86_64 architecture Ubuntu LTS (Xenial Xerus), Ubuntu LTS (Bionic Beaver), CentOS, EulerOS, RHEL, Kylin, NewStart CGS Linux, NeoKylin, openEuler, Unity Operating System (UOS), Oracle Linux (OL), Huawei Cloud Euler (HCE), openEuler 23.09 Edge</li> <li>• Armv7i (Arm32) architecture Raspbian GNU/Linux (stretch)</li> <li>• AArch64 (Arm64) architecture Ubuntu LTS (Xenial Xerus), Ubuntu LTS (Bionic Beaver), CentOS, EulerOS, RHEL, Kylin, NewStart CGS Linux, NeoKylin, openEuler, Unity Operating System (UOS), Oracle Linux (OL), Huawei Cloud Euler (HCE), openEuler 23.09 Edge</li> </ul> <p><b>NOTE</b> The openEuler 23.09 Edge operating system is recommended for edge computing scenarios.</p>
Memory	<p>More than 256 MB of memory is recommended as 128 MB of memory is required to run the edge software.</p>
CPU	<p>≥ 1 core</p>
Hard disk	<p>≥ 1 GB</p>
GPU (optional)	<p>The GPU models on the same edge node must be the same.</p> <p><b>NOTE</b> Currently, NVIDIA Tesla GPUs such as P4, P40, and T4 are supported.</p> <p>If an edge node is equipped with GPUs, you can choose not to enable its GPUs when registering it on IEF.</p> <p>If you choose to enable GPUs of an edge node, the GPU driver has to be installed on the edge node before you can manage it on IEF.</p> <p>Currently, only x86-based GPU nodes can be managed by IEF.</p>

Item	Specifications
NPU (optional)	<p>Ascend AI processors</p> <p><b>NOTE</b> Currently, edge nodes integrated with Ascend Processors are supported, such as Atlas 300 inference cards, and Atlas 800 inference servers. Supported NPU specifications include Ascend 310P, 310B, Ascend 310P-share, and virtualization partition NPUs..</p> <p>If you choose to enable NPUs of an edge node, ensure that the NPU driver has been installed on it. Currently, Ascend 310 supports only firmware versions 1.3.x.x and 1.32.x.x, for example, 1.3.2.B893. You can run the <b>npu-smi info</b> command to view your firmware version.The NPU driver version must be 22.0.4 or later. You can go to the driver path, for example, <b>/usr/local/Ascend/driver</b>, and run the <b>cat version.info</b> command to view your driver version. If the driver is not installed, contact the device manufacturer for assistance.</p>
Container engine	<p>The Docker version must be later than 17.06. If Docker 1.23 or later is used, set the docker cgroupfs version to 1. Docker HTTP API v2 is not supported.</p> <p>(However, Docker 18.09.0 is not recommended as it has a serious bug. For details, see <a href="https://github.com/docker/for-linux/issues/543">https://github.com/docker/for-linux/issues/543</a>. If this version has been installed, upgrade it at the earliest possible opportunity. )</p> <p><b>NOTICE</b> After Docker is installed, configure the Docker process to start at host startup. This configuration prevents system exceptions caused by Docker startup failures after the host is restarted.</p> <p>Docker <b>Cgroup Driver</b> must be set to <b>cgroupfs</b>. For details, see <a href="#">How Do I Set Docker Cgroup Driver After Installing Docker on an Edge Node?</a>.</p>
Glibc	The Glibc version must be later than 2.17.
Port	Edge nodes require port 8883, which is the listening port of the built-in MQTT broker on edge nodes. Ensure that this port works properly.
Time synchronization	The time on an edge node must be consistent with the UTC time. Otherwise, the monitoring data and logs of the edge node may be inaccurate. You can select an NTP server for time synchronization. For details, see <a href="#">How Do I Synchronize Time with the NTP Server?</a>

## Configuring the Edge Node Environment

**Step 1** Log in to an edge node as a user with sudo permissions.

**Step 2** Configure the GPU driver.

If your edge node has been equipped with a GPU, install and configure the GPU driver on the edge node. For details, see [Installing and Configuring a GPU Driver](#).

**Step 3** Configure the NPU driver.

If your edge node uses Ascend AI processors, ensure that the corresponding driver has been installed.

**Step 4** Install Docker on the edge node and check the Docker status.

The Docker version must be later than 17.06. Docker 18.06.3 is recommended. However, Docker 18.09.0 is not recommended as it has a serious bug. If you have used this version, upgrade it at the earliest possible opportunity.

After Docker installation is complete, run **docker -v** to check whether Docker was installed properly. If the following information is displayed, Docker was installed properly.

```
# docker -v  
Docker version 19.03.12, build 48a66213fee
```

**Step 5** Configure firewall rules for the edge node.

Check the firewall status on the edge node.

```
systemctl status firewalld  
firewall-cmd --state
```

In the command output, **not running** indicates that the firewall is disabled and **running** indicates that the firewall is enabled.

If the firewall is enabled, enable port 8883 or disable the firewall.

- To enable port 8883, run the following commands:  
firewall-cmd --add-port=8883/tcp --permanent  
systemctl restart firewalld
- To disable the firewall, run the following commands:  
systemctl disable firewalld  
systemctl stop firewalld

----End

## Caution

To improve host security, you are advised to harden the OS of the edge node by performing the following operations:

1. Set strong passwords for all OS accounts (including administrators and common users), database accounts, and application (web) system management accounts. Each password must contain at least 12 characters.
2. Do not run applications using the administrator account. Disallow applications (such as webs) to use the database administrator account to interact with databases. Configure security groups and open only necessary ports to the public network. Protect the service web console ports and LAN internal communication ports from being exposed to the public network. Disable high-risk ports (such as the SSH port), allow limited source IP addresses to access the ports, or use the O&M channel established based on VPNs or bastion hosts.

3. Periodically back up service data remotely to prevent data loss caused by intrusions.
4. Periodically detect security vulnerabilities in the system and software, update system security patches in a timely manner, and upgrade the software to the latest official version.
5. Download and install the software from official channels. For the software downloaded from non-official channels, use antivirus software to scan it before running.

If you use Huawei Cloud Elastic Cloud Server (ECS), perform the following operations:

1. Set the host login mode to key login.
2. Use Huawei Cloud [Host Security Service \(HSS\)](#) for in-depth defense.

### 4.1.3 Registering an Edge Node

Registering an edge node is the process of creating an edge node on IEF and obtaining the node configuration file and EdgeCore installer.

#### Constraints

- NPUs on the same node support only the same partition specification.
- You cannot perform virtualization partition on the NPU of an IEF-managed edge node by simply upgrading the NPU plug-in to v2.1.0. Uninstall and reinstall the node, manually partition the NPU, and then manage the node on IEF again.
- The NPU driver version must be later than 22.0. You can go to the driver path (for example, `/usr/local/Ascend/driver`) and run the `cat version.info` command to view your driver version.
- If you enable the NPU sharing mode, you need to set the container sharing mode for all chips on the NPU.
- NPU virtualization partition is not supported if the D310P-share is used.
- The healthy chips displayed refer to physical chips. If NPU virtualization partition or sharing is used, the number of chips after virtualization partition or sharing is displayed.

#### Registering an Edge Node

- Step 1** Log in to the IEF console, and click **Switch Instance** on the **Dashboard** page to select a platinum service instance.
- Step 2** In the navigation pane, choose **Managed Resources > Edge Nodes**. Then, click **Register Edge Node** in the upper right corner.
- Step 3** Configure basic edge node information.

**Figure 4-2** Basic edge node information (1)

Service Instance 👑 platinum

Node Name

Description  0/255

Tags    
You can add 20 more tags.

---

**ⓘ** If you need to deploy NPU or GPU applications, please switch the AI accelerator card type according to your choice. [Edge Node Specifications Requirements](#)

AI Accelerator Card Not installed Huawei AI accelerator card NVIDIA GPU

Bind Device

Device Names	Device to Node Relationship	Remarks	Operation
⊕ Add Device			

---

Docker ?

Listening Address ? NIC IP

tls://	<input type="text" value="lo"/>	:	<input type="text" value="8883"/>	<input type="button" value="🗑"/>
tls://	<input type="text" value="docker0"/>	:	<input type="text" value="8883"/>	<input type="button" value="🗑"/>

⊕ Add Listening Address

- **Node Name:** name of an edge node. The name can contain 1 to 64 characters, including letters, digits, hyphens (-), and underscores (\_).
- **Description:** Optional. Enter the description of the edge node.
- **Tags:** Tags can be used to classify resources, facilitating resource management.
- **AI Accelerator Card**  
Ascend AI accelerator card: supports edge nodes that support Ascend processors and NPU specifications of Ascend 310P and Ascend 310P virtualization partition. If you need to use Ascend 310, 310B, 910, 310P, 310P-share, or 310P-1c, first select the **AI accelerator card** and then select the NPU type.

AI accelerator cards support the NPU types listed in the following table:

**Table 4-2** NPU types

Type	Description
Ascend 310	Ascend 310 chips
Ascend 310B	Ascend 310B chips
Ascend 910	Ascend 910 chips
Ascend 310P	Ascend 310P chips

Type	Description
Ascend 310P-share	Ascend 310P chips that support multi-container sharing
Ascend 310P-1c	Ascend 310P chips after virtualization partition based on the template <b>vir01</b>
Ascend 310P-2c	Ascend 310P chips after virtualization partition based on the template <b>vir02</b>
Ascend 310P-2c.1cpu	Ascend 310P chips after virtualization partition based on the template <b>vir02_1c</b>
Ascend 310P-4c	Ascend 310P chips after virtualization partition based on the template <b>vir04</b>
Ascend 310P-4c.3cpu	Ascend 310P chips after virtualization partition based on the template <b>vir04_3c</b>

**NVIDIA GPU:** Select this option if your edge node is equipped with an NVIDIA GPU.

**Not installed:** Select this option if your edge node does not use any AI accelerator card.

 **NOTE**

If you select **NVIDIA GPU** but your edge node is not equipped with any NVIDIA GPU, the edge node cannot be managed by IEF.

If your edge node has been equipped with a GPU, install and configure the GPU driver on the edge node before you can manage it on IEF. For details, see [Installing and Configuring a GPU Driver](#).

- **Bind Device:** Bind an end device to the edge node. If you have not registered an end device, register one by referring to [End Device Management](#). If you do not bind an end device in this step, you can bind one after the edge node is registered.
- **Docker:** If you enable Docker, containerized applications can be deployed.
- **Listening Address**

The address on which the built-in MQTT broker of the edge node listens. It is used for edge-cloud messaging. For details about edge-cloud messaging, see [Device Twin Working Principles](#) and [Edge-Cloud Message Overview](#).

By default, local NICs lo (localhost) and docker0 are listened to. You can specify an NIC name or IP address to set the NIC to be listened to. Multiple NICs or IP addresses can be added.



**Figure 4-3** Basic edge node information (2)

System Logs Application Logs

Set Maximum Log Size and Logs Saved to values that will prevent logs from taking up too much space and impacting the node.

Maximum Log Size (MB)

Log Rotation

Logs Saved

Save Logs to AOM

Currently, system and application logs can be collected from edge nodes.

- **System Logs:** logs generated by the IEF software (such as edge-core, edge-logger, and edge-monitor) installed on edge nodes.
- **Application Logs:** logs generated by applications deployed on edge nodes.

Set the following parameters for both system and application logs:

- **Maximum Log Size (MB):** maximum size of a log file, in MB. The default value is **50**. The value range is 10–1000. If a log file reaches the specified size, the system compresses the log file and dumps it to a specified directory on the host.
  - System logs are stored in the `/var/IEF/sys/log/` directory on edge nodes and will be dumped to AOM when the maximum log size is reached.
  - Application logs include the standard output of containers and the logs mounted to `/var/IEF/app/log` on the edge node. These logs will be dumped to AOM when the maximum log size is reached.
- **Log Rotation:** log dump interval. You can select **Daily**, **Weekly**, **Monthly**, or **Yearly**. Log files will be dumped when either **Maximum Log Size (MB)** or **Log Rotation** is reached.
- **Logs Saved:** maximum number of logs being retained at a specified period. The default value is **5**. The value range is 1–10. Once this number is reached, when new log files are dumped in, old log files will be deleted on a first-in first-out (FIFO) basis.
- **Save Logs to AOM**

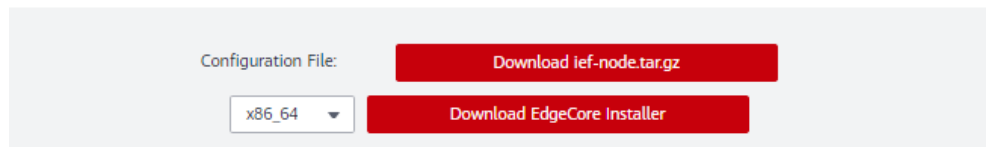
You can enable or disable log uploading to AOM. If it is enabled, you can view the logs in AOM. For details, see [Viewing Logs on AOM](#).

**Step 4** Select **I have read and agree to the Huawei Cloud Service Level Agreement**. Click **Register** in the lower right corner.

**Registration using a certificate:** Download the configuration file and EdgeCore Installer, which will be used in [Managing an Edge Node](#).

**Figure 4-4** Downloading the configuration file and EdgeCore Installer

Download the following configuration files and software to complete the creation.  
Download the certificate right now because you will not be able to retrieve it afterwards.



1. Click **Download *Edge node name.tar.gz***.
2. Select the CPU architecture of your edge node, and click **Download EdgeCore Installer**.

**Step 5** In the lower right corner, select **I've finished downloading** and click **Finish**.

The edge node is in the **Unconnected** state because the EdgeCore Installer downloaded in [Registering an Edge Node](#) has not been installed. For details, see [Managing an Edge Node](#).

**Figure 4-5** Unconnected edge node

<input type="checkbox"/>	Name/ID	Node Status	Host Name/Network	Instances (Normal/All)	Created	Edge Software...	Node Tags	Node Source	Operation
<input type="checkbox"/>	ief-node 569d31f4-83de-4699-abc	Unconnected Installation Guide	--	0/0	Jul 20, 2021 09:20:23 GMT...	--	--	Custom node	Delete   More

----End

## Follow-up Operation

After the registration is complete, you need to manage the edge node. For details, see [Managing an Edge Node](#).

### 4.1.4 Managing an Edge Node

Managing an edge node is the process of using the EdgeCore installer and configuration file downloaded in [Registering an Edge Node](#) to install EdgeCore on the edge node. In this way, the edge node can be connected to and managed by IEF.

When you manage your edge node on IEF for the first time, IEF automatically pushes the latest EdgeCore. For example, if three EdgeCore versions 2.51.0, 2.52.0, and 2.53.0 are available, IEF will push the latest version 2.53.0.

#### NOTE

The edge nodes registered on IEF have one-to-one relationships with the actual devices. The EdgeCore installer and configuration file of an edge node can be installed on only one actual device.

## Prerequisites

- You have prepared a node that meets the specified environment requirements. For details, see [Configuring the Edge Node Environment](#).
- You have registered a node on IEF and obtained the node configuration file and EdgeCore installer. For details, see [Registering an Edge Node](#).

## Managing an Edge Node

**Step 1** Log in to an edge node as a user with sudo permissions.

**Step 2** Upload the EdgeCore installer and configuration file downloaded in [Registering an Edge Node](#) to a specified directory on the edge node, for example, `/home`, and go to the directory.

**Step 3** Run the following command to decompress the EdgeCore installer to the `/opt` directory:

```
sudo tar -zxvf edge-installer_1.0.0_x86_64.tar.gz -C /opt
```

Replace `edge-installer_1.0.0_x86_64.tar.gz` with the name of the EdgeCore installer package downloaded in [Registering an Edge Node](#).

**Step 4** Run the following command to decompress the configuration file to the `opt/IEF/Cert` directory. If the edge node to be managed is registered using a token, skip this step.

```
sudo mkdir -p /opt/IEF/Cert; sudo tar -zxvf Edge node name.tar.gz -C /opt/IEF/Cert
```

Replace `Edge node name.tar.gz` with the name of the configuration file downloaded in [Registering an Edge Node](#).

**Step 5** Run one of the following commands to manage the edge node:

- For an edge node registered using a certificate:

```
cd /opt/edge-installer; sudo ./installer -op=install
```

**Step 6** Verify whether the edge node is managed successfully.

1. Log in to the IEF console, and click **Switch Instance** on the **Dashboard** page to select a platinum service instance.
2. In the navigation pane, choose **Managed Resources > Edge Nodes**.
3. Check the edge node status. If the status is **Running**, the edge node has been managed by IEF.

**Figure 4-6** Checking the edge node status

Name/ID	Node Status	Host Name/Network	Instances (Normal/A...	Created	Edge Software Version
ief-node 7092ad14-adee-4a09-b969-1505	Running	eth0:192.168.0.230	0/0	Jul 20, 2021 09:3...	2.52.0 Upgradeable

----End

### NOTICE

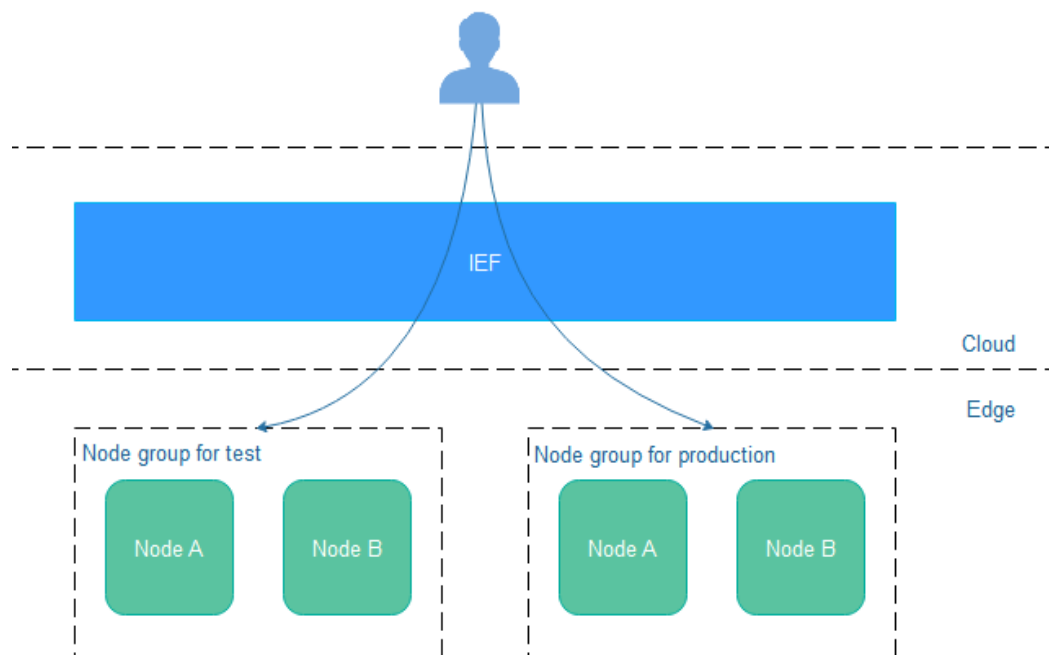
After the edge node is managed by IEF, do not delete the `/opt` directory on the edge node, or you will need to register the edge node and enable IEF to manage it again.

## 4.1.5 Edge Node Groups

An edge node group can divide nodes and facilitate deployment of applications on nodes in the group. You can perform the following operations on node groups:

- Add nodes to node groups, and remove nodes and end devices from node groups.
- Allocate security certificates for accessing nodes in a node group and applications deployed in the node group.
- Automatically schedule applications onto nodes in a node group based on the resource usage of the nodes.
- Reschedule an application onto other normal nodes when a node in a node group is faulty.
- Specify affinity and anti-affinity scheduling rules for applications and nodes.

Figure 4-7 Node groups




### Creating an Edge Node Group

- Step 1** Log in to the IEF console, and click **Switch Instance** on the **Dashboard** page to select a platinum service instance.
- Step 2** In the navigation pane, choose **Managed Resources > Edge Node Groups**. Then, click **Create Edge Node Group** in the upper right corner.
- Step 3** Set the name and description of the node group, and select the edge nodes to be added to the node group, and click **Create**.

After a node group is created, you can still add or remove nodes.

**Figure 4-8** Creating an edge node group

Service Instance  test\_2

★ Name

Description   
0/255

Tags    
You can add 20 more tags.

---

Node Group Type  IEF  IEC

Edge Node

----End

## Deploying an Application in a Specified Node Group

You can specify a node group where an application is to be deployed. The application will be deployed to the suitable node in the group based on the resources of the nodes in the group. For details on how to deploy applications, see [Containerized Applications](#).

**Figure 4-9** Creating a containerized application

\* Deployment Object Manually specify Automatically schedule

---

Edge Node Group Select Group The application is automatically scheduled onto edge nodes in the group based on the resource usage.

Fault Handling Policy Migrate Not migrate Time Window (s)

**i** If the proportion of unavailable nodes in the node group is greater than 0.55, the automatic migration stops.

Scheduling Policies Simple Scheduling Policy

---

Advanced Settings ▼

- **Fault Handling Policy:** Indicates whether the application instance is rescheduled onto another available node in the edge node group when the edge node that runs the application instance is unavailable.

**NOTE**

If more than 55% of nodes is unavailable in a node group, automatic migration will stop.

- **Time Window (s):** Indicates the duration of a fault before rescheduling is triggered.

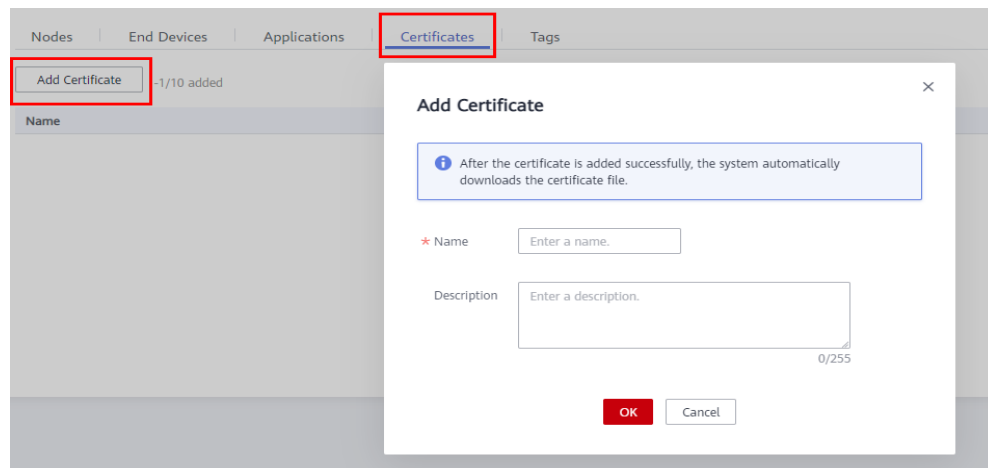
## Adding a Node Group Certificate

Applications deployed in a node group may be scheduled onto any node in the node group. End devices may also access any node in the node group. The certificates added on the node details page cannot implement access to the MQTT broker on nodes from applications. Therefore, a node group certificate is required.

**NOTE**

The validity period of a certificate is 5 years.

Click the node group name to go to the details page. On the **Certificates** tab page, add a certificate.

**Figure 4-10** Adding a node group certificate

## 4.1.6 Upgrading an Edge Node

### Context

IEF releases new versions of EdgeCore irregularly. You can upgrade the EdgeCore software installed on your edge node as required.

### Version Support Policy

IEF only maintains the edge node software versions released within one year. Therefore, you are advised to upgrade your edge nodes at least once a year.

### Version Upgrade Rule

During edge node upgrade, IEF automatically selects the EdgeCore of the latest version.

For example, if EdgeCore 2.22.0, 2.23.0, and 2.24.0 are available and the EdgeCore version on your edge node is 2.12.0, IEF will push the EdgeCore 2.24.0 to your edge node.

### Precautions

- IEF does not proactively upgrade EdgeCore on your edge nodes. You are advised to upgrade the node manually in the time window with the minimum impact on your services.
- Upgrading a version within the maintenance period will not interrupt your applications running on the edge node. However, if message routing is used, services may be temporarily affected.
- Upgrading a version beyond the maintenance period may temporarily interrupt services due to container restart.
- Do not change node configurations during the node upgrade, such as restarting Docker, installing or uninstalling the GPU/NPU driver, upgrading the OS kernel, or modifying network configurations, which may result in node upgrade failures.

## Procedure

**Step 1** Log in to an edge node, and configure firewall rules.

Check the firewall status on the edge node.

```
systemctl status firewalld
firewall-cmd --state
```

In the command output, **not running** indicates that the firewall is disabled and **running** indicates that the firewall is enabled.

If the firewall is enabled, enable port 8883 or disable the firewall.

- To enable port 8883, run the following commands:  
firewall-cmd --add-port=8883/tcp --permanent  
systemctl restart firewalld
- To disable the firewall, run the following commands:  
systemctl disable firewalld  
systemctl stop firewalld

**Step 2** Log in to the IEF console, and click **Switch Instance** on the **Dashboard** page to select a platinum service instance.

**Step 3** In the navigation pane, choose **Managed Resources > Edge Nodes**.

**Step 4** View the **Edge Software Version** column to check whether an upgrade can be performed.

Only edge nodes in the **Running** state can be upgraded.

- If **Upgradeable** is displayed in the **Edge Software Version** column, you can perform an upgrade.
- If **Upgradeable** is not displayed, check whether the edge node is in the **Running** state. If the edge node is in the **Running** state and **Upgradeable** is not displayed, the edge node already runs the latest EdgeCore.

**Figure 4-11** Checking whether an edge node can be upgraded

Name/ID	Node	Host Name/Netw...	Instances (No...	Created	Edge Soft...	Node Tags	Node S...	Operation
ief-node 7092ad14-adee-4a09-b969-1505bbdeco	Running	eth0:192.168.0.230	0/0	Jul 20, 2021 09:33:26 GMT...	2.52.0 Upgradeable	--	Custom...	Delete   More

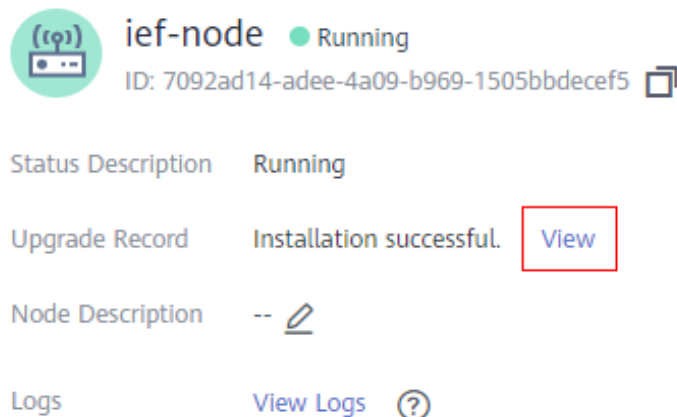
**Step 5** Choose **More > Upgrade**.

**Figure 4-12** Upgrading an edge node

Name/ID	Node	Host Name/Netw...	Instances (No...	Created	Edge Soft...	Node Tags	Node S...	Operation
ief-node 7092ad14-adee-4a09-b969-1505bbdeco	Running	eth0:192.168.0.230	0/0	Jul 20, 2021 09:33:26 GMT...	2.52.0 Upgradeable	--	Custom...	Delete   More Enable Disable View Monitoring Upgrade

**Step 6** Click the node name to go to the node details page, and view the upgrade record.



**Figure 4-13** Upgrade record

----End

## 4.1.7 Logs, Monitoring, and Alarms

### Log Description

Enabling logging for an edge node on the IEF console will upload the system and application logs of the edge node to Application Operations Management (AOM).

- **System Logs:** logs generated by the IEF software (such as edge-core, edge-logger, and edge-monitor) installed on edge nodes.
- **Application Logs:** logs generated by applications deployed on edge nodes.
  - Edge nodes will upload logs in the `/var/IEF/app/log` directory to AOM through IEF. You can mount the `/var/IEF/app/log/{appName}` directory into the container when creating an application. For details, see [hostPath: used for mounting a directory of the host into the container](#). You can view logs on AOM by `{appName}`.
  - Edge nodes will also upload container logs in `{{DOCKER_ROOT_DIR}}/containers/{containerID}/{containerID}-json.log` to AOM. You can run the `docker info` command to query the value of `DOCKER_ROOT_DIR`. `containerID` indicates the container ID.

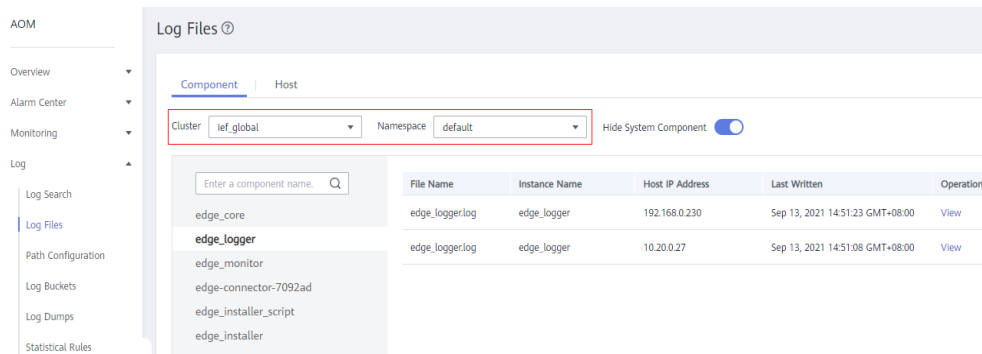
### Viewing Logs on AOM

**Step 1** Log in to the AOM console.

**Step 2** In the navigation pane, choose **Log > Log Files**, and click the **Component** tab.

**Step 3** On the displayed page, select cluster **ief\_global** and namespace **default**.

**Figure 4-14** Selecting a cluster and a namespace



**Step 4** Search for logs by application name, and click **View** in the row where the log file resides to view detailed logs.

----End

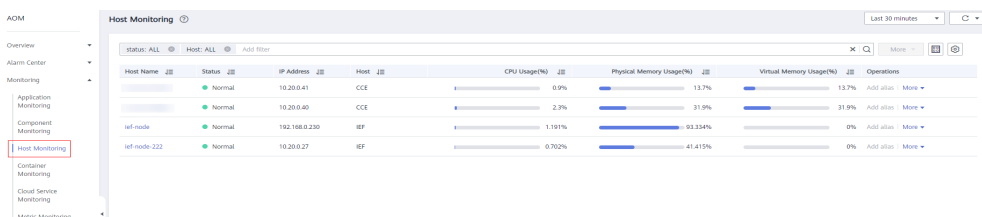
## Viewing Node Monitoring Information on AOM

You can view the node monitoring information on AOM.

**Step 1** Log in to the AOM console.

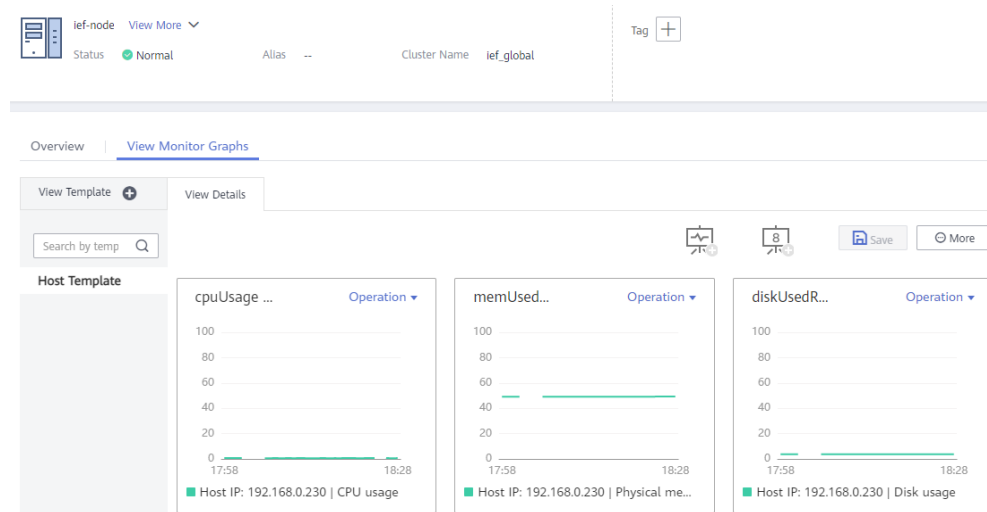
**Step 2** Click the name of the node whose monitoring information is to be viewed.

**Figure 4-15** Selecting a monitored node



**Step 3** On the **View Monitor Graphs** tab page, view the resource usage of the node, such as the CPU usage and memory usage.

**Figure 4-16** Viewing monitoring information



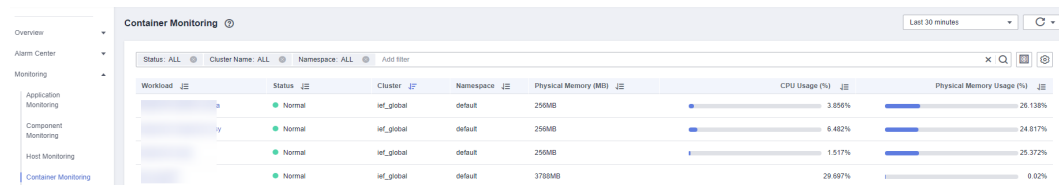
----End

## Viewing Container Monitoring Information on AOM

You can view the monitoring information about the containerized applications deployed on edge nodes on AOM.

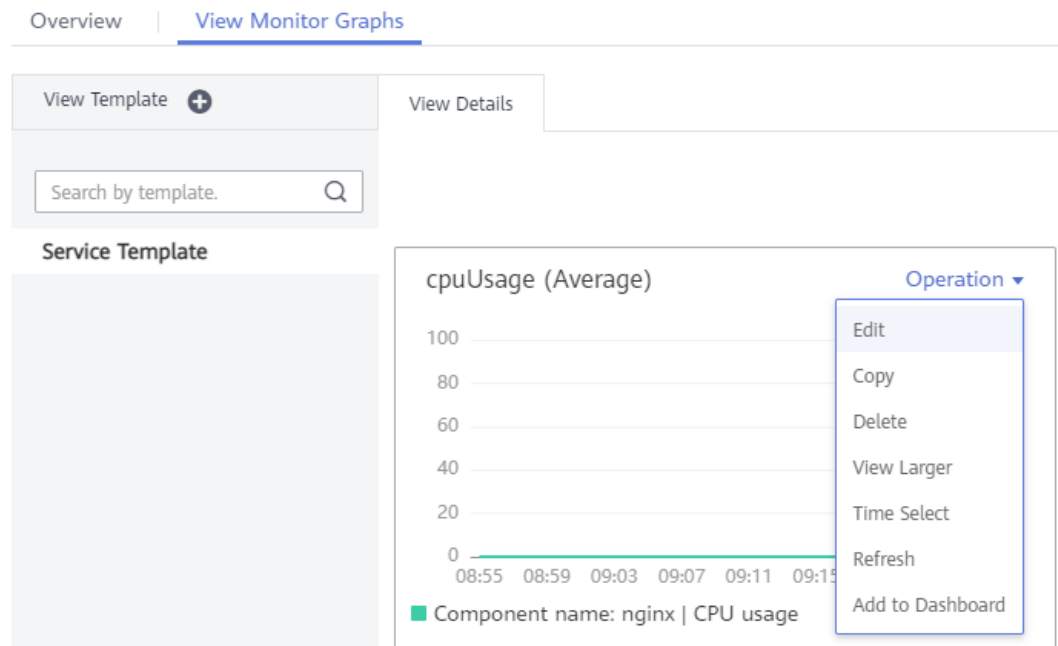
- Step 1** Log in to the AOM console.
- Step 2** Select the workload whose monitoring information is to be viewed.

**Figure 4-17** Selecting a workload



- Step 3** On the **View Monitor Graphs** tab page, view the monitoring metrics of the container, such as the CPU usage and memory usage.

**Figure 4-18** Viewing monitoring information



----End

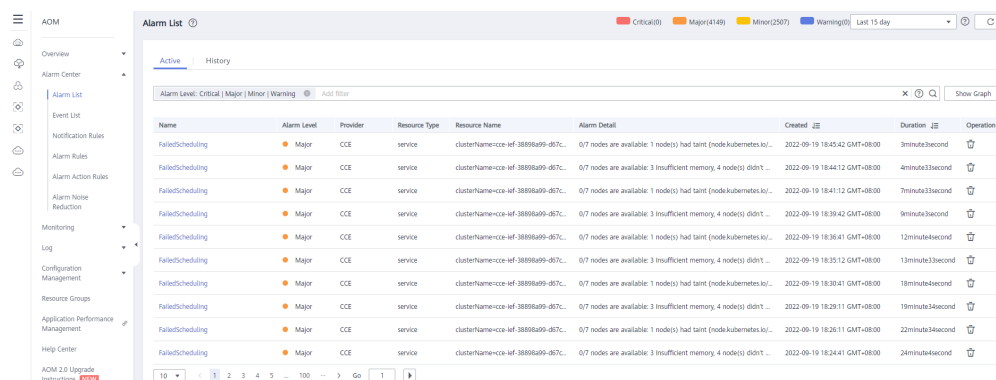
## Alarms Preset on IEF

IEF comes with seven types of alarms preconfigured for each edge node. These alarms will be automatically reported to AOM.

Alarm Name	Trigger	Clearance Condition	Severity
Container Engine Is Abnormal	Docker has been enabled on an edge node, and the Docker information fails to be queried.	Docker is running properly, and EdgeCore can obtain the Docker information.	Critical
Application Liveness Probe Is Abnormal	A liveness probe has been configured for the application, and the probe detects an exception.	The probe detects that the container is normal.	Major
Failed to Obtain GPU Resources	GPU resources could not be obtained during GPU application deployment.	GPU resources are obtained.	Critical
Failed to Obtain GPU Information	GPU has been enabled on an edge node, and the GPU information fails to be queried.	The GPU information is successfully queried.	Critical

Alarm Name	Trigger	Clearance Condition	Severity
Invalid AK/SK	EdgeHub has distributed 10 consecutive temporary AK/SK pairs and detects that the AK/SK has expired or is in abnormal state.	EdgeHub successfully distributes the temporary AK/SK.	Major
Application Restarted	The application container restarts unexpectedly.	This alarm does not need to be cleared.	Minor
NIC Bound to the Container Is Faulty	The NIC bound to the container is faulty.	The NIC bound to the container becomes normal.	Critical

Figure 4-19 Viewing alarms



## Setting Alarms on AOM

You can create alarm rules on AOM to monitor metrics of edge nodes. For details, see [Creating Threshold Rules](#).

## Reporting User-Defined Alarms to AOM

IEF can report customized alarms from edge nodes to AOM. To be specific, after the MQTT client publishes alarm information to the MQTT broker, IEF will automatically report the alarms to AOM.

For details, see [Alarm Reporting](#) and [Alarm Clearance](#).

## 4.1.8 Installing and Configuring a GPU Driver

### Context

For an edge node that uses GPUs, you need to install and configure the GPU driver before managing the edge node on IEF.

Currently, IEF supports NVIDIA Tesla GPUs such as P4, P40 and T4, and the GPU drivers that match CUDA Toolkit 8.0 to 10.0.

## Procedure

**Step 1** Install the GPU driver.

1. Download the GPU driver. The recommended driver link is as follows:

[https://www.nvidia.com/content/DriverDownload-March2009/confirmation.php?url=/tesla/440.33.01/NVIDIA-Linux-x86\\_64-440.33.01.run&lang=us&type=Tesla](https://www.nvidia.com/content/DriverDownload-March2009/confirmation.php?url=/tesla/440.33.01/NVIDIA-Linux-x86_64-440.33.01.run&lang=us&type=Tesla)

2. Run the following command to install the GPU driver:

```
bash NVIDIA-Linux-x86_64-440.33.01.run
```

3. Run the following command to check the GPU driver installation status:

```
nvidia-smi
```

**Step 2** Log in to the edge node as user **root**.

**Step 3** Run the following command:

```
nvidia-modprobe -c0 -u
```

**Step 4** Create directories.

```
mkdir -p /var/IEF/nvidia/drivers /var/IEF/nvidia/bin /var/IEF/nvidia/lib64
```

**Step 5** Copy GPU driver files to the directories.

- For CentOS, run the following commands in sequence to copy the driver files:  

```
cp /lib/modules/{Kernel version of the current environment}/kernel/drivers/video/nvi* /var/IEF/nvidia/drivers/  
cp /usr/bin/nvidia-* /var/IEF/nvidia/bin/  
cp -rd /usr/lib64/libcuda* /var/IEF/nvidia/lib64/  
cp -rd /usr/lib64/libEG* /var/IEF/nvidia/lib64/  
cp -rd /usr/lib64/libGL* /var/IEF/nvidia/lib64/  
cp -rd /usr/lib64/libnv* /var/IEF/nvidia/lib64/  
cp -rd /usr/lib64/libOpen* /var/IEF/nvidia/lib64/  
cp -rd /usr/lib64/libvdpau_nvidia* /var/IEF/nvidia/lib64/  
cp -rd /usr/lib64/vdpau /var/IEF/nvidia/lib64/
```
- For Ubuntu, run the following commands in sequence to copy the driver files:  

```
cp /lib/modules/{Kernel version of the current environment}/kernel/drivers/video/nvi* /var/IEF/nvidia/drivers/  
cp /usr/bin/nvidia-* /var/IEF/nvidia/bin/  
cp -rd /usr/lib/x86_64-linux-gnu/libcuda* /var/IEF/nvidia/lib64/  
cp -rd /usr/lib/x86_64-linux-gnu/libEG* /var/IEF/nvidia/lib64/  
cp -rd /usr/lib/x86_64-linux-gnu/libGL* /var/IEF/nvidia/lib64/  
cp -rd /usr/lib/x86_64-linux-gnu/libnv* /var/IEF/nvidia/lib64/  
cp -rd /usr/lib/x86_64-linux-gnu/libOpen* /var/IEF/nvidia/lib64/  
cp -rd /usr/lib/x86_64-linux-gnu/libvdpau_nvidia* /var/IEF/nvidia/lib64/
```

```
cp -rd /usr/lib/x86_64-linux-gnu/vdpau /var/IEF/nvidia/lib64/
```

You can run the **uname -r** command to view the kernel version of the current environment. The following is an example. Replace the kernel version with the actual value.

```
# uname -r  
3.10.0-514.e17.x86_64
```

**Step 6** Run the following command to change the directory permissions:

```
chmod -R 755 /var/IEF
```

----End

## 4.1.9 EdgeCore Configuration Management

### Scenario

IEF allows you to manage EdgeCore configuration parameters, through which you can let EdgeCore work under your very requirements.

### Procedure

**Step 1** Run the following command on the edge node to modify the EdgeCore configuration and save the modification:

```
vi /opt/IEF/Edge-core/conf/edge.yaml
```

The following table describes the parameters that can be configured.

**Table 4-3** Parameter description

Component	Parameter	Description	Value
edge-core	interface-name	NIC name	Default value: <b>eth0</b>
	internal-server	Listening address of the built-in MQTT broker	tls://lo:8883,tls://docker0:8883
	mage-gc-high-threshold	Percentage of the disk usage that triggers image garbage collection	Default value: <b>80</b>
	image-gc-low-threshold	Target percentage of the disk usage after releasing resources with image garbage collection	Default value: <b>40</b>
	swr-url	Proxy address for pulling an image	Default value: ""

**Step 2** After the configuration is modified, restart EdgeCore.

```
systemctl restart edgecore
```

----End

## 4.1.10 Deleting an Edge Node

### Prerequisites

Before deleting an edge node, you need to unbind the end devices and delete applications and certificates on the edge node.

### Procedure

**Step 1** Run the **sudo** command to log in to the edge node.

**Step 2** Run the following commands to uninstall the software and configuration files on the managed nodes:

```
cd /opt/edge-installer; sudo ./installer -op=uninstall
```

**Step 3** Log in to the IEF console.

**Step 4** In the navigation pane, choose **Managed Resources > Edge Nodes**.

**Step 5** Locate the row which contains the edge node to be deleted. In the **Operation** column, choose **More > Delete**.

**Step 6** Delete the node as prompted.

----End

## 4.2 End Device Management

### 4.2.1 End Devices and Device Twins

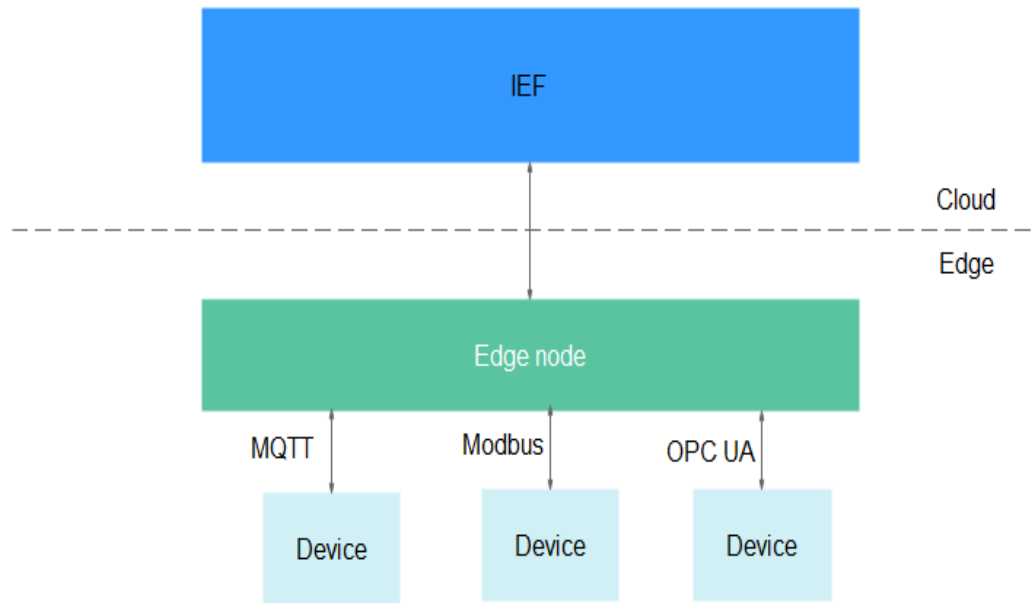
#### End Device

End devices can be as small as a sensor or controller or as large as a smart camera or computer numerical control (CNC) machine tool.

These devices can be connected to IEF through edge nodes by using MQTT. After end devices are connected to IEF, you can manage them on IEF in a unified manner.



**Figure 4-20** End device management

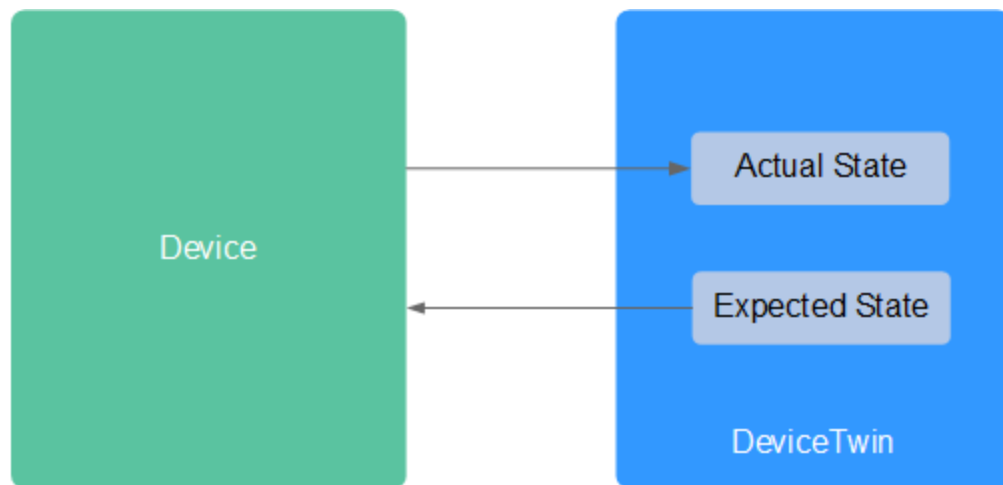


## Device Twins

An end device usually has two types of data:

- Metadata that does not change, which includes the serial number, asset identifier, MAC address, and other information about the device. This type of data can be called static properties or device properties.
- Dynamic device data, which includes dedicated real-time device data for a specific scenario, for example, the on and off states of a light. This type of data can be called twin properties.

DeviceTwin has the same features as physical devices and facilitates better communication between end devices and applications. The commands sent by an application reach DeviceTwin (an internal component on an edge node). Then, DeviceTwin updates the device status according to **Expected State** set by the application. In addition, the end device reports its **Actual State** in real time. DeviceTwin records both **Actual State** and **Expected State** of the end device. In this way, the status of an offline device can be synchronized when it comes back online.

**Figure 4-21** DeviceTwin

On the IEF console, you can register an end device and bind it to an edge node. After the binding, the device and twin properties will be stored on the edge node. Applications deployed on the edge node can obtain device and twin properties, and modify **Expected State** and **Actual State** recorded in DeviceTwin. In addition, IEF synchronizes twin properties between the cloud and edge. If a conflict occurs, the properties at the edge are used.

For details about the edge-cloud synergy for the end device status, see [Device Twin Working Principles](#).

## Procedure

To enable IEF to manage and control an end device, perform the following steps:

1. Define a device template (including device and twin properties).
2. Use the template to register a device.  
You can also register a device without using a template.
3. Bind the device to an edge node.
4. Perform related operations, such as monitoring the device status.

### 4.2.2 Device Templates

In an edge computing scenario, there are a large number of end devices. You can define a template for end devices of the same type and use it to register the devices on IEF. For example, you can create a device template named **Camera**. After you use this template to register end devices, these devices will use all properties defined in the template. In this way, you do not need to keep repeating the same configuration.

IEF allows you to define templates based on access protocols.

## Creating a Device Template

- Step 1** Log in to the IEF console, and click **Switch Instance** on the **Dashboard** page to select a platinum service instance.

**Step 2** In the navigation pane, choose **Managed Resources > End Devices**. Then, click **Create Device Template** in the upper right corner.

**Step 3** Enter a template name, select an access protocol, and set template description, device properties, twin properties, and device tags.

**Figure 4-22** Creating a device template

The screenshot shows a web form for creating a device template. At the top, it indicates the 'Service Instance' is 'test\_2'. The 'Name' field is required and contains the placeholder 'Enter a template name'. The 'Access Protocol' is set to 'Custom'. The 'Description' field is a large text area with the placeholder 'Enter a description.' and a character count of '0/255'. Below this is a 'Device Property' section with a warning: 'The information entered here will be viewable. Do not enter sensitive information.' It features a table with columns: Key, Type, Value, Optional, and Operation. There is an 'Add Property' button below the table. The 'Tags' section has two input fields: 'Tag key' and 'Tag value', with a note 'You can add 20 more tags.'

- **Name:** name of a device template.
- **Access Protocol:** IEF supports **MQTT**.
- **Device Property:**  
Properties are defined as key-value pairs. Enter a property name and value, and select a type.  
Metadata that does not change, such as serial numbers, asset identifiers, and MAC addresses, is defined in templates as device properties.
- **Twin Property:**  
Dynamic device data, such as dedicated real-time device data for a specific scenario, is set as twin properties. For example, the on and off states of a light are real-time data.
  - **MQTT:** Twin properties are defined as key-value pairs. Enter a property name and value, and select a type.

#### NOTICE

IEF does not provide any encryption or decryption tools, and does not sense your device attribute values. If the device attribute values are encrypted, you need to decrypt them.

- **Tags:** Tags are used to classify end devices. You can quickly search for desired devices by tag. Tags also facilitate end device management by category.

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

----End

## 4.2.3 End Devices

These devices can be connected to IEF through edge nodes by using MQTT. After end devices are connected to IEF, you can manage them on IEF in a unified manner.

### Registering an End Device

**Step 1** Log in to the IEF console, and click **Switch Instance** on the **Dashboard** page to select a platinum service instance.

**Step 2** In the navigation pane, choose **Managed Resources > End Devices**. Then, click **Register End Device** in the upper right corner.

**Step 3** Set device parameters.

- **Name:** name of an end device.
- **ID:** You can customize a device ID or enable IEF to automatically generate an ID. If you want to customize a device ID, select **Custom device ID** and enter one.
- **Access Protocol:** Choose **MQTT**.
- **Device Configuration:** Select an existing device template to add properties for the device automatically. The template defines device properties, twin properties, and device tags. You can also manually add device properties and tags without using a template. The meanings of the properties and tags manually added are the same as those defined in the template. For details, see [Device Templates](#).

Note: The device template can be selected only when the access protocol of it is the same as that selected during end device registration.

**Step 4** Click **Register**.

----End

### Follow-up Procedure

You can bind end devices to edge nodes. For details, see [Binding an End Device to an Edge Node](#).

## 4.2.4 Binding an End Device to an Edge Node

An edge node can be bound to multiple end devices, but each end device can be bound to only one edge node. By binding an end device to an edge node, you can deploy applications on the edge node so that you can manage the device and monitor its status on IEF.

### Binding an Edge Node

**Step 1** Log in to the IEF console, and click **Switch Instance** on the **Dashboard** page to select a platinum service instance.

- Step 2** In the navigation pane, choose **Managed Resources > End Devices**.
- Step 3** Click **Bind Node** in the row where the end device is located.
- Step 4** Enter the relationship between the device and the node, select the edge node to be bound, and click **OK**.

----End

## Binding an End Device on the Edge Node Details Page

You can also bind an end device on the details page of an edge node.

- Step 1** Log in to the IEF console.
- Step 2** In the navigation pane, choose **Managed Resources > Edge Nodes**. Then, click an edge node name.
- Step 3** On the edge node details page, click the **End Devices** tab and then click **Bind**.
- Step 4** In the dialog box that is displayed, select the device to be bound, enter the relationship between the device and the node, and click **OK**.

### NOTE

Nodes and devices are dependent on each other. Only nodes with Modbus plug-ins installed can be bound to Modbus devices.

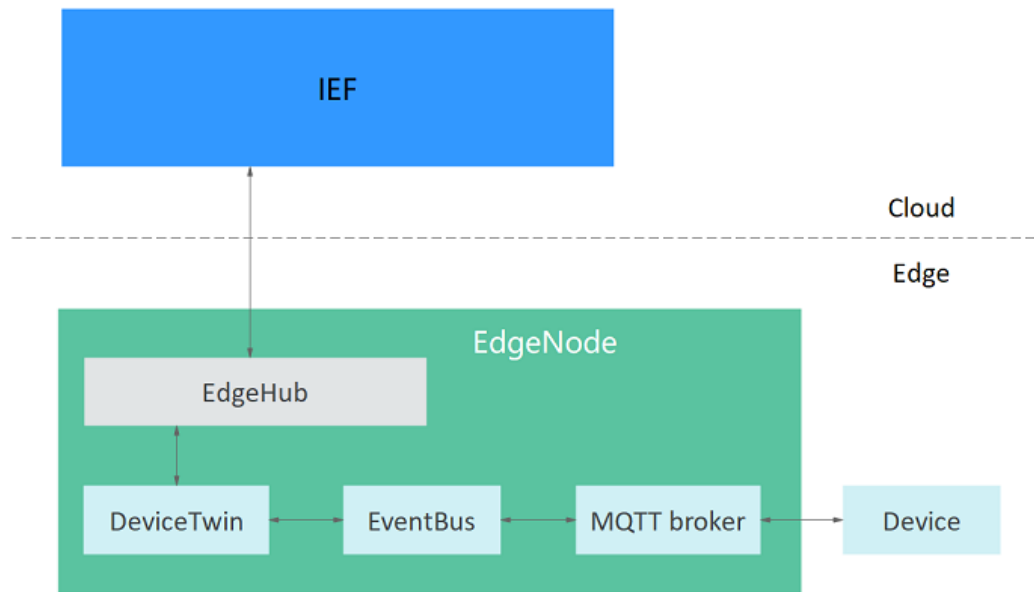
----End

## 4.2.5 Device Twin Working Principles

After an edge node is managed, Edge Agent is installed on the edge node. Edge Agent provides the following components for end device management:

- EdgeHub: a WebSocket client, which provides functions such as synchronizing cloud resource updates and reporting edge node and end device data to the cloud.
- DeviceTwin: stores end device status and synchronizes the end device status to the cloud.
- EventBus: a client that interacts with the MQTT server and provides the functions of subscribing to and publishing messages for other components.
- MQTT broker: an MQTT server.

**Figure 4-23** End device management



DeviceTwin plays an important role during the communication among end devices, edge nodes, and IEF. It maintains the dynamic device data, including the dedicated real-time data of devices in a specific scenario, such as the on/off status of lights.

DeviceTwin has the same features as physical devices and facilitates better communication between end devices and applications. The commands sent by an application reach DeviceTwin. Then, DeviceTwin updates the device status according to **Expected State** set by the application. In addition, the end device reports its **Actual State** in real time. DeviceTwin records both **Actual State** and **Expected State** of the end device. In this way, the status of an offline device can be synchronized when it comes back online.

The following is an example of a device twin:

```

{
  "device": {
    "id": "989e4fc8-9f24-44d7-9f9c-a0bd3bfb1949",
    "description": "my home light",
    "name": "light",
    "state": "online",
    "twin": {
      "powerstatus": {
        "expected": {
          "value": "ON"
        },
        "actual": {
          "value": "OFF"
        },
        "metadata": {
          "type": "string"
        }
      }
    }
  }
}
    
```

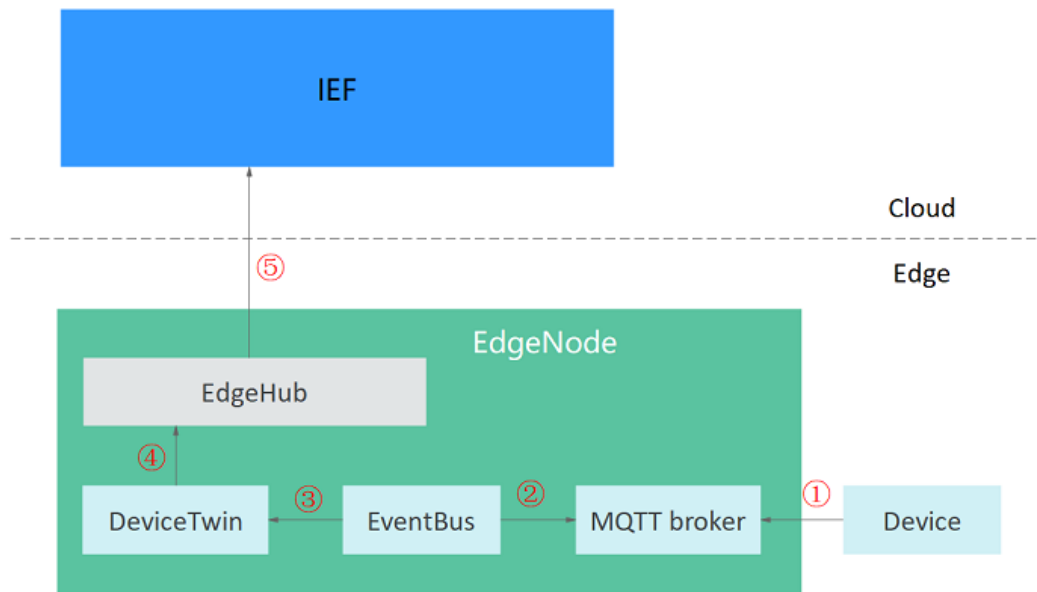
In this example, a twin property **powerstatus** is defined for the device named **light**, in which the expected value is **ON** and the actual value is **OFF**.

The following describes how this end device communicates with the edge node and IEF.

## Reporting the Actual Device Status to the Cloud

**Figure 4-24** shows the process for an end device to report its actual status to the cloud.

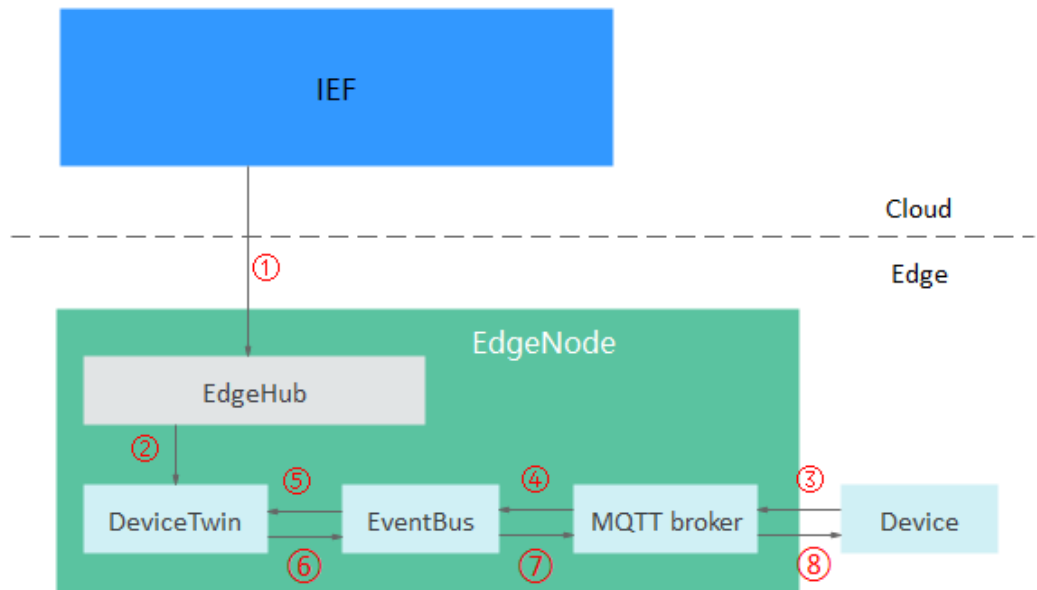
**Figure 4-24** Reporting the device status



1. The end device reports its actual status to the MQTT broker in real time.
2. EventBus receives a subscription message from the MQTT broker. The message contains the actual status of the end device.
3. EventBus sends the actual device status to DeviceTwin. Then, DeviceTwin stores the actual device status on the edge node.
4. DeviceTwin synchronizes the actual status to EdgeHub (a WebSocket client).
5. EdgeHub sends a message to IEF.

## Modifying Twin Properties in the Cloud to Control Device Status

Figure 4-25 Modifying the device status



1. Modify the twin properties of the end device on IEF. Then, IEF sends the expected status of the end device to EdgeHub on the edge node.
2. EdgeHub sends a message containing the expected device status to DeviceTwin. Then, DeviceTwin stores the expected device status on the edge node.
3. The end device sends a message to the MQTT broker in real time to query the expected device status.
4. EventBus receives the message from the MQTT broker.
5. EventBus queries the expected device status based on the message.
6. DeviceTwin sends the expected device status to EventBus.
7. EventBus sends the expected device status to the MQTT broker.
8. The end device receives a subscription message from the MQTT broker and adjusts its status to the expected status.

### 4.2.6 Migrating Device Data to the Cloud

#### MQTT Broker

End devices can communicate with IEF using the MQTT protocol. You can also control end devices through pub/sub messaging.

An edge node has a **built-in MQTT broker**. The built-in MQTT broker uses port 8883 to communicate with end devices. End devices can communicate with the built-in MQTT broker only after they pass security authentication. For details, see [Performing Security Authentication Using Certificate](#).

Edge nodes also support **external MQTT brokers**. You can install an MQTT broker (for example, [Mosquitto](#), an open-source MQTT broker) on an edge node. Then devices can communicate with the edge node over port 1883.



 NOTE

If an external MQTT broker is used, ensure that the communication port of the external MQTT broker is available.

## MQTT Topics

The MQTT broker forwards messages between end devices and edge nodes to enable the end devices to communicate with the nodes and IEF. By default, the MQTT broker provides message topics described in [Table 4-4](#). Device status can be reported and controlled through pub/sub messaging.

After an application is compiled, you can deploy the application to an edge node through IEF. For details, see [Containerized Application Management](#).

**Table 4-4** Default topics provided by IEF

Name	Type	Topic	Description
<a href="#">Device Twin Update</a>	Subscribe	<code>\$hw/events/device/{device_id}/twin/update/document</code>	This topic is used to subscribe to device twin updates. It reflects the differences before and after a device twin update.
<a href="#">Device Twin Delta</a>	Subscribe	<code>\$hw/events/device/{device_id}/twin/update/delta</code>	This topic is used to subscribe to device twin delta events. When a device twin changes, the twin properties whose actual values are different from expected values are returned.
<a href="#">Device Member Update</a>	Subscribe	<code>\$hw/events/node/{node_id}/membership/updated</code>	This topic is used to subscribe to end device binding updates.
<a href="#">Device Property Update</a>	Subscribe	<code>\$hw/events/device/{device_id}/updated</code>	This topic is used to subscribe to end device property updates.
<a href="#">Device Member Acquisition</a>	Publish	<code>\$hw/events/node/{node_id}/membership/get</code>	This topic is used to subscribe to end device bindings.
<a href="#">Device Member Acquisition Result</a>	Subscribe	<code>\$hw/events/node/{node_id}/membership/get/result</code>	This topic is used to subscribe to the result of obtaining information about end device members.
<a href="#">Device Twin Acquisition</a>	Publish	<code>\$hw/events/device/{device_id}/twin/get</code>	This topic is used to publish the request for obtaining device twins.

Name	Type	Topic	Description
<b>Device Twin Acquisition Result</b>	Subscribe	\$hw/events/device/{device_id}/twin/get/result	This topic is used to subscribe to the result of obtaining device twins.
<b>Device Twin Modification</b>	Publish	\$hw/events/device/{device_id}/twin/update	This topic is used to publish device twin modifications.
<b>Device Twin Modification Result</b>	Subscribe	\$hw/events/device/{device_id}/twin/update/result	This topic is used to subscribe to the result of modifying device twins.
<b>Encryption Data Request</b>	Publish	\$hw/{project_id}/encryptdatas/{encryptdata_name}/properties/{properties_name}/decrypt	This topic is used to publish the request for obtaining encryption data.
<b>Encryption Data Acquisition</b>	Subscribe	\$hw/{project_id}/encryptdatas/{encryptdata_name}/properties/{properties_name}/plaintext	This topic is used to subscribe to encryption data.
<b>Alarm Reporting</b>	Publish	\$hw/alarm/{appname}/add	This topic is used to report alarms to AOM.
<b>Alarm Clearance</b>	Publish	\$hw/alarm/{appname}/clear	This topic is used to clear alarms in AOM.
<b>Custom Topics</b>	Publish	{project_id}/nodes/{node_id}/user/{custom_topic}	Topics customized based on actual requirements. You can publish end device data to a custom topic in the MQTT broker of an edge node. IEF then forwards the device data to a DIS stream or an API Gateway address. Then, you can extract the data for processing and analysis.

The following describes how to obtain end device data at the edge, receive control messages from the cloud, and report end device data to the cloud. For details

about the sample code for MQTT to send and receive messages, see [Go-Language Code Sample](#) and [Java-Language Code Sample](#).

## Obtaining End Devices Associated with an Edge Node

**Step 1** Send a request to the topic [Device Member Acquisition](#).

**Topic:** \$hw/events/node/{node\_id}/membership/get

**Payload:** {"event\_id": "Custom ID"}

For example:

```
$hw/events/node/{node_id}/membership/get  
{"event_id":""}
```

**Step 2** Subscribe to the topic [Device Member Acquisition Result](#).

**Topic:** \$hw/events/node/{node\_id}/membership/get/result

The following is an example of the returned result:

```
{  
  "event_id": "",  
  "timestamp": 1554986455386,  
  "devices": [  
    {  
      "id": "2144773f-13f1-43f5-af07-51991d4fd064",  
      "name": "equipmentA",  
      "state": "unknown",  
      "attributes": {  
        "name": {  
          "value": "a",  
          "optional": true,  
          "metadata": {  
            "type": "string"  
          }  
        }  
      }  
    }  
  ]  
}
```

----End

## Obtaining Device Twins

**Step 1** Send a request to the topic used to [Device Twin Acquisition](#).

**Topic:** \$hw/events/device/{device\_id}/twin/get

**Payload:** {"event\_id": "Custom ID"}

**Step 2** Subscribe to the topic [Device Twin Acquisition Result](#).

**Topic:** \$hw/events/device/{device\_id}/twin/get/result

The following is an example of the returned result:

```
{  
  "event_id": "",  
  "timestamp": 1554988425592,  
  "twin": {  
    "humidity": {  
      "expected": {  
        "value": "0",
```

```

        "metadata": {
          "timestamp": 1554988419529
        }
      },
      "optional": true,
      "metadata": {
        "type": "int"
      }
    },
    "temperature": {
      "expected": {
        "value": "0",
        "metadata": {
          "timestamp": 1554988419529
        }
      }
    },
    "optional": true,
    "metadata": {
      "type": "int"
    }
  }
}

```

----End

## Listening to Device Twin Events

After **Obtaining End Devices Associated with an Edge Node** and **Obtaining Device Twins** are performed, the ID of the end device bound to the node is obtained. Then, the events of the end device can be listened to.

Update the device twin by setting the expected value in the cloud to control the end device.

For example, a device has two twin properties: humidity and temperature.

**Figure 4-26** Twin properties

Overview   <u>Device Twins</u>   Tags			
+ Add Twin Property			
Name <small>⌵</small>	Type	Expected Value	Created
humidity	int	9	Jul 13, 2020 09:40:38 GMT+08:00
temperature	int	0	Jul 13, 2020 09:40:58 GMT+08:00

Change the value of twin property **humidity** on the **Device Twins** tab page from **9** to **10**. After the twin property is changed, the device receives the following two events:

- **Device twin change event:** includes the twin information before and after the change.
- **Device twin delta event:** includes the device twin details and the delta information about the inconsistency between the expected and actual values of twin properties.

The device can receive the two events only after it has subscribed to the two topics.

**Step 1** Subscribe to the topic **Device Twin Update**.**Topic:** \$hw/events/device/{device\_id}/twin/update/document

The update message received at the edge is as follows:

```
{
  "event_id": "0f921313-4074-46a2-96f6-aac610721059",
  "timestamp": 1555313685831,
  "twin": {
    "humidity": {
      "last": {
        "expected": {
          "value": "9",
          "metadata": {
            "timestamp": 1555313665978
          }
        },
        "optional": true,
        "metadata": {
          "type": "int"
        }
      },
      "current": {
        "expected": {
          "value": "10",
          "metadata": {
            "timestamp": 1555313685831
          }
        },
        "optional": true,
        "metadata": {
          "type": "int"
        }
      }
    },
    "temperature": {
      "last": {
        "expected": {
          "value": "0",
          "metadata": {
            "timestamp": 1555313665978
          }
        },
        "actual": {
          "value": "2",
          "metadata": {
            "timestamp": 1555299457284
          }
        },
        "optional": true,
        "metadata": {
          "type": "int"
        }
      },
      "current": {
        "expected": {
          "value": "0",
          "metadata": {
            "timestamp": 1555313685831
          }
        },
        "actual": {
          "value": "2",
          "metadata": {
            "timestamp": 1555299457284
          }
        },
        "optional": true,
        "metadata": {

```

```

        "type": "int"
      }
    }
  }
}

```

**Step 2** Subscribe to the topic **Device Twin Delta**.

**Topic:** \$hw/events/device/{device\_id}/twin/update/delta

The update message received at the edge is as follows:

```

{
  "event_id": "60fb5baf-d4ad-47b0-a21e-8b57b52d0978",
  "timestamp": 1555313685837,
  "twin": {
    "humidity": {
      "expected": {
        "value": "10",
        "metadata": {
          "timestamp": 1555313685831
        }
      },
      "optional": true,
      "metadata": {
        "type": "int"
      }
    },
    "temperature": {
      "expected": {
        "value": "0",
        "metadata": {
          "timestamp": 1555313685831
        }
      },
      "actual": {
        "value": "2",
        "metadata": {
          "timestamp": 1555299457284
        }
      },
      "optional": true,
      "metadata": {
        "type": "int"
      }
    }
  },
  "delta": {
    "humidity": "10",
    "temperature": "0"
  }
}

```

----End

## Reporting Actual Values of Device Properties

**Step 1** Publish a twin update event for an end device.

**Topic:** \$hw/events/device/{device\_id}/twin/update

**Payload:** {"event\_id":"","timestamp":0,"twin":{"Property":{"actual":{"value":"Actual value"}}}}

For example:

```

{
  "event_id": "",

```

```

"timestamp": 0,
"twin": {
  "temperature": {
    "actual": {
      "value": "2"
    }
  }
}
}

```

After the event is published, you can find that the actual value of the device twin has changed in the cloud, as shown in [Figure 4-27](#).

**Figure 4-27** Updated twin properties

Name	Type	Expected Value	Created	Actual Value
humidity	int	9	Jul 13, 2020 09:40:38 GMT+08:00	
temperature	int	0	Jul 13, 2020 09:49:22 GMT+08:00	2

**Step 2** Subscribe to the topic **Device Twin Modification Result** at the edge. The edge node then can receive the result of the device twin update event.

**Topic:** \$hw/events/device/{device\_id}/twin/update/result

The update result is as follows:

```

{
  "event_id": "",
  "timestamp": 1554992093859,
  "twin": {
    "temperature": {
      "actual": {
        "value": "2",
        "metadata": {
          "timestamp": 1554992093859
        }
      }
    },
    "optional": true,
    "metadata": {
      "type": "int"
    }
  }
}
}

```

----End

## 4.2.7 Performing Security Authentication Using Certificate

### Scenario

By default, the built-in MQTT broker enables the port for Transport Layer Security (TLS) authentication. A client can access the MQTT broker only when it has a certificate.

End devices and applications can use the certificates added on the node details page for security authentication.

 NOTE

Applications deployed in a node group may be scheduled onto any node in the node group. End devices may also access any node in the node group. The certificates added on the node details page cannot implement access to the MQTT broker on nodes from applications. Therefore, a node group certificate is required. For details on how to add a node group certificate, see [Adding a Node Group Certificate](#).

## Constraints

- Certificates are bound to edge nodes. The certificates applied for on an edge node can be used only to access the MQTT broker of this edge node. If these certificates are used to access the MQTT broker of another edge node, the authentication will fail.
- A maximum of 10 certificates can be applied for an edge node.
- The validity period of a certificate is 5 years.
- There are constraints on using MQTT.

**Table 4-5** MQTT constraints

Description	Restriction
Supported MQTT version	3.1.1
Differences from the standard MQTT protocol	<ul style="list-style-type: none"><li>• Quality of Service (QoS) 0 is supported.</li><li>• Topic customization is supported.</li><li>• QoS 1 and QoS 2 are not supported.</li><li>• <b>will</b> and <b>retain</b> messages are not supported.</li></ul>
MQTTS security levels	TCP channel basic + TLS protocol (TLS v1.2)

## Applying for a Certificate

 NOTE

The validity period of a certificate is 5 years.

- Step 1** Log in to the IEF console, and click **Switch Instance** on the **Dashboard** page to select a platinum service instance.
- Step 2** In the navigation pane, choose **Managed Resources > Edge Nodes**.
- Step 3** Click an edge node name. The edge node details page is displayed.
- Step 4** Click the **Certificates** tab, and click **Add Certificate**.
- Step 5** Enter a certificate name and click **OK**.

After the certificate is added, the system will automatically download the certificate file. Keep the certificate file secure.



**Figure 4-28** Adding a certificate

**Add Certificate** ×

**i** After the certificate is added successfully, the system automatically downloads the certificate file.

\* Name

Description  0/255

**OK**

----End

## Using a Certificate

A certificate is used for authentication when an end device communicates with the MQTT broker.

[Go-Language Code Sample](#) and [Java-Language Code Sample](#) illustrate how to use certificates for authentication.

### NOTE

- The client does not need to verify the server certificate. In other words, one-way authentication is required.
- Port 8883 of the built-in MQTT broker is enabled by default.
- In the Go-language code sample, the MQTT client references [github.com/eclipse/paho.mqtt.golang](https://github.com/eclipse/paho.mqtt.golang) (an open-source library).
- The MQTT client is required to process disconnection events and reestablish connections to improve the connection reliability.

## Go-Language Code Sample

```
package main

import (
    "crypto/tls"
    "crypto/x509"
    "fmt"
    "math/rand"
    "sync"
    "time"

    MQTT "github.com/eclipse/paho.mqtt.golang"
)
```

```
func main() {
    subClient := InitMqttClient(onSubConnectionLost)
    pubClient := InitMqttClient(onPubConnectionLost)

    wait := sync.WaitGroup{}
    wait.Add(1)

    go func() {
        for {
            time.Sleep(1*time.Second)
            pubClient.Publish("topic", 0, false, "hello world")
        }
    }()

    subClient.Subscribe("topic", 0, onReceived)

    wait.Wait()
}

func InitMqttClient(onConnectionLost MQTT.ConnectionLostHandler) MQTT.Client {
    pool := x509.NewCertPool()
    cert, err := tls.LoadX509KeyPair("/tmp/example_cert.crt", "/tmp/example_cert.key")
    if err != nil {
        panic(err)
    }

    tlsConfig := &tls.Config{
        RootCAs: pool,
        Certificates: []tls.Certificate{cert},
        // One-way authentication, that is, the client does not verify the server certificate.
        InsecureSkipVerify: true,
    }
    // Use the TLS or SSL protocol to connect to port 8883.
    opts := MQTT.NewClientOptions().AddBroker("tls://127.0.0.1:8883").SetClientID(fmt.Sprintf("%f", rand.Float64()))
    opts.SetTLSConfig(tlsConfig)
    opts.OnConnect = onConnect
    opts.AutoReconnect = false
    // Callback function. It is triggered immediately after the client is disconnected from the server.
    opts.OnConnectionLost = onConnectionLost
    client := MQTT.NewClient(opts)
    loopConnect(client)
    return client
}

func onReceived(client MQTT.Client, message MQTT.Message) {
    fmt.Printf("Receive topic: %s, payload: %s \n", message.Topic(), string(message.Payload()))
}

// The reconnection mechanism is triggered after the subscribe client is disconnected from the server.
func onSubConnectionLost(client MQTT.Client, err error) {
    fmt.Println("on sub connect lost, try to reconnect")
    loopConnect(client)
    client.Subscribe("topic", 0, onReceived)
}

// The reconnection mechanism is triggered after the publish client is disconnected from the server.
func onPubConnectionLost(client MQTT.Client, err error) {
    fmt.Println("on pub connect lost, try to reconnect")
    loopConnect(client)
}

func onConnect(client MQTT.Client) {
    fmt.Println("on connect")
}

func loopConnect(client MQTT.Client) {
    for {
```

```
token := client.Connect()
if rs, err := CheckClientToken(token); !rs {
    fmt.Printf("connect error: %s\n", err.Error())
} else {
    break
}
time.Sleep(1 * time.Second)
}
}

func CheckClientToken(token MQTT.Token) (bool, error) {
    if token.Wait() && token.Error() != nil {
        return false, token.Error()
    }
    return true, nil
}
```

## Java-Language Code Sample

The format of an MqttClientDemo.java file is as follows:

```
/*
Description: A java demo of MQTT message sending and receiving. You need to create an edge node and
download the client certificate.
*/

package com.example.demo;

import javax.net.ssl.SSLSocketFactory;

import org.eclipse.paho.client.mqttv3.IMqttDeliveryToken;
import org.eclipse.paho.client.mqttv3.MqttCallback;
import org.eclipse.paho.client.mqttv3.MqttClient;
import org.eclipse.paho.client.mqttv3.MqttConnectOptions;
import org.eclipse.paho.client.mqttv3.MqttException;
import org.eclipse.paho.client.mqttv3.MqttMessage;
import org.eclipse.paho.client.mqttv3.persist.MemoryPersistence;

/*
* MQTT demo shows how the client connects to the MQTT broker of the edge node to send and receive
messages and how SSL security authentication is performed on the connection. The demo illustrates the
following flows:
* 1. The MQTT subscribe client receives MQTT messages.
* 2. The MQTT publish client sends MQTT messages.
*/

public class MqttClientDemo {
    private static int QOS_TYPE = 2;
    //MQTT server address
    private static final String MQTT_HOST = "ssl://x.x.x.x:8883";
    //MQTT publish client ID
    private static final String MQTT_PUB_CLIENT_ID = "pub_client_1";
    //MQTT subscribe client ID
    private static final String MQTT_SUB_CLIENT_ID = "sub_client_1";
    //MQTT channel subscription topic
    private static final String TOPIC = "/hello";
    //Paths of the SSL certificate used for MQTT client connection
    public static final String CLIENT_CERT_FILE_PATH = "example_cert.crt";
    public static final String CLIENT_KEY_FILE_PATH = "example_cert.key";
    //MQTT client connection timeout interval (s)
    public static final int TIME_OUT_INTERVAL = 10;
    //Interval at which the MQTT client sends a heartbeat message, in seconds
    public static final int HEART_TIME_INTERVAL = 20;
    //Interval at which the MQTT client retries upon disconnection, in milliseconds
    public static final int RECONNECT_INTERVAL = 10000;
    //Interval at which the MQTT client sends a message, in seconds
    public static final int PUBLISH_MSG_INTERVAL = 3000;

    //MQTT client
    private MqttClient mqttClient;
```

```
//MQTT client ID.
private String clientId;
//MQTT client connection options
private MqttConnectOptions connOpts;
//Initialized MQTT client has not subscribed to any topic.
private boolean isSubscribe = false;

public MqttClientDemo(String id) throws MqttException {
    setClientId(id);
    initMqttClient();
    initCallback();
    initConnectOptions();
    connectMqtt();
}

/*****
 * Sending messages
 * @param message Message to be sent
 * @throws MqttException
 *****/
public void publishMessage(String message) throws MqttException {
    MqttMessage mqttMessage = new MqttMessage(message.getBytes());
    mqttMessage.setQos(QOS_TYPE);
    mqttMessage.setRetained(false);
    mqttClient.publish(TOPIC, mqttMessage);
    System.out.println(String.format("MQTT Client[%s] publish message[%s]", clientId, message));
}

/*****
 * Subscribing to topics
 * @throws MqttException
 *****/
public void subscribeTopic() throws MqttException {
    int[] Qos = {QOS_TYPE};
    String[] topics = {TOPIC};
    mqttClient.subscribe(topics, Qos);
    isSubscribe = true;
}

/*****
 * Starting the thread to periodically send MQTT messages
 * @throws MqttException
 *****/
public void startPublishMessage() {
    new Thread() {
        @Override
        public void run() {
            while (true) {
                try {
                    Thread.sleep(PUBLISH_MSG_INTERVAL);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                try {
                    publishMessage("hello world!");
                } catch (MqttException e) {
                    System.out.println(String.format("MQTT client[%s] publish message error,errorMsg[%s]",
clientId, e.getMessage()));
                }
            }
        }
    }.start();
}

/*****
 * Initializing the MQTT client
 * @throws MqttException Abnormal connection
 *****/
private void initMqttClient() throws MqttException {
```

```
MemoryPersistence persistence = new MemoryPersistence();
mqttClient = new MqttClient(MQTT_HOST, clientId, persistence);
}

/*****
 * Initializing connection options
 * @throws MqttException Abnormal connection
 *****/
private void initConnectOptions() {
    connOpts = new MqttConnectOptions();
    // Specify whether to clear the session. If the value is false, the server retains the client connection
records. If the value is true, the client connects to the server as a new client.
    connOpts.setCleanSession(true);
    connOpts.setHttpsHostnameVerificationEnabled(false);
    // Set the timeout interval, in seconds.
    connOpts.setConnectionTimeout(TIME_OUT_INTERVAL);
    // Set the interval at which a session heartbeat message is sent, in seconds. The server sends a
message to the client every 1.5x20 seconds to check whether the client is online. However, this method
does not provide the reconnection mechanism.
    connOpts.setKeepAliveInterval(HEART_TIME_INTERVAL);
    SSLSocketFactory factory = null;
    try {
        factory = SslUtil.getSocketFactory(CLIENT_CERT_FILE_PATH, CLIENT_KEY_FILE_PATH);
    } catch (Exception e) {
        e.printStackTrace();
    }
    // TLS connection configuration
    connOpts.setSocketFactory(factory);
}

/*****
 * Initiate an MQTT connect request.
 * @throws MqttException Abnormal connection
 *****/
private void connectMqtt() throws MqttException {
    mqttClient.connect(connOpts);
    System.out.println(String.format("MQTT client[%s] is connected,the connOptions: \n%s", clientId,
connOpts.toString()));
}

/*****
 * Set the callback API.
 * @throws MqttException Abnormal connection
 *****/
private void initCallback() {
    mqttClient.setCallback(new MqttMessageCallback());
}

private void setClientId(String id) {
    clientId = id;
}

/*****
 * MQTT client reconnection function. This function is called to check whether a topic has been
subscribed to. If yes, the topic will be re-subscribed to.
 * @throws MqttException
 *****/
private void rconnectMqtt() throws MqttException {
    connectMqtt();
    if (isSubscribe) {
        subscribeTopic();
    }
}

/*****
 * After the MQTT client subscribes to topics, the MQTT client receives messages through the callback
API if the MQTT channel has data.
 * @version V1.0
 *****/
```

```
private class MqttMessageCallback implements MqttCallback {

    @Override
    public void connectionLost(Throwable cause) {
        System.out.println(String.format("MQTT Client[%s] connect lost,Retry in 10 seconds,info[%s]",
clientId, cause.getMessage()));
        while (!mqttClient.isConnected()) {
            try {
                Thread.sleep(RECONNECT_INTERVAL);
                System.out.println(String.format("MQTT Client[%s] reconnect ....", clientId));
                reconnectMqtt();
            } catch (Exception e) {
                continue;
            }
        }
    }

    @Override
    public void messageArrived(String topic, MqttMessage mqttMessage) {
        String message = new String(mqttMessage.getPayload());
        System.out.println(String.format("MQTT Client[%s] receive message[%s] from topic[%s]", clientId,
message, topic));
    }

    @Override
    public void deliveryComplete(IMqttDeliveryToken iMqttDeliveryToken) {

    }
}

public static void main(String[] args) throws MqttException {
    try {
        //Subscribe to the MQTT channel.
        MqttClientDemo mqttsubClientDemo = new
MqttClientDemo(MqttClientDemo.MQTT_SUB_CLIENT_ID);
        mqttsubClientDemo.subscribeTopic();
        //Send hello world to the MQTT channel.
        MqttClientDemo mqttpubClientDemo = new
MqttClientDemo(MqttClientDemo.MQTT_PUB_CLIENT_ID);
        mqttpubClientDemo.startPublishMessage();
    } catch (MqttException e) {
        System.out.println(String.format("program start error,errorMessage[%s]", e.getMessage()));
    }
}
}
```

The format of an SSLUtil.java file is as follows:

```
/*
Description: SSL utility class. Load the client SSL certificate configuration and ignore server certificate
verification.
*/

package com.example.demo;

import java.io.ByteArrayInputStream;
import java.io.InputStreamReader;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.security.KeyPair;
import java.security.KeyStore;
import java.security.Security;
import java.security.cert.Certificate;
import java.security.cert.CertificateException;
import java.security.cert.X509Certificate;

import javax.net.ssl.KeyManagerFactory;
import javax.net.ssl.SSLContext;
```

```
import javax.net.ssl.SSLSocketFactory;
import javax.net.ssl.TrustManager;
import javax.net.ssl.X509TrustManager;

import org.bouncycastle.jce.provider.BouncyCastleProvider;
import org.bouncycastle.openssl.PEMReader;
import org.bouncycastle.openssl.PasswordFinder;

public class SslUtil {

    /*****
     * Verify and obtain the SSLSocketFactory.
     *****/
    public static SSLSocketFactory getSocketFactory(final String crtFile, final String keyFile) throws Exception
    {
        Security.addProvider(new BouncyCastleProvider());

        // 1. Load the client certificate.
        PEMReader reader_client =
            new PEMReader(new InputStreamReader(new
                ByteArrayInputStream(Files.readAllBytes(Paths.get(crtFile))));
            X509Certificate cert = (X509Certificate) reader_client.readObject();
            reader_client.close();

        // 2. Load the client key.
        reader_client = new PEMReader(
            new InputStreamReader(new ByteArrayInputStream(Files.readAllBytes(Paths.get(keyFile))),
            new PasswordFinder() {
                @Override
                public char[] getPassword() {
                    return null;
                }
            }
        );

        // 3. Send the client key and certificate to the server for identity authentication.
        KeyStore ks = KeyStore.getInstance(KeyStore.getDefaultType());
        ks.load(null, null);
        ks.setCertificateEntry("certificate", cert);
        ks.setKeyEntry("private-key", ((KeyPair) reader_client.readObject()).getPrivate(), "".toCharArray(), new
        Certificate[] {cert});
        KeyManagerFactory kmf =
        KeyManagerFactory.getInstance(KeyManagerFactory.getDefaultAlgorithm());
        kmf.init(ks, "".toCharArray());

        // 4. Create a socket factory.
        SSLContext context = SSLContext.getInstance("TLSv1.2");
        TrustManager[] tms = new TrustManager[1];
        TrustManager miTM = new TrustAllManager();
        tms[0] = miTM;
        context.init(kmf.getKeyManagers(), tms, null);

        reader_client.close();

        return context.getSocketFactory();
    }

    /*****
     * Ignore server certificate verification.
     *****/
    static class TrustAllManager implements TrustManager, X509TrustManager {
        @Override
        public X509Certificate[] getAcceptedIssuers() {
            return null;
        }

        @Override
```

```
public void checkServerTrusted(X509Certificate[] certs, String authType)
    throws CertificateException {
}

public boolean isServerTrusted(X509Certificate[] certs) {
    return true;
}

public boolean isClientTrusted(X509Certificate[] certs) {
    return true;
}

@Override
public void checkClientTrusted(X509Certificate[] certs, String authType)
    throws CertificateException {
}
}
```

The format of a pom.xml file is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.example</groupId>
    <artifactId>mqtt.example</artifactId>
    <version>1.0-SNAPSHOT</version>
    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <configuration>
                    <source>7</source>
                    <target>7</target>
                </configuration>
            </plugin>
        </plugins>
    </build>
    <dependencies>
        <!-- https://mvnrepository.com/artifact/org.eclipse.paho/org.eclipse.paho.client.mqttv3 -->
        <dependency>
            <groupId>org.eclipse.paho</groupId>
            <artifactId>org.eclipse.paho.client.mqttv3</artifactId>
            <version>1.2.1</version>
        </dependency>
        <!-- https://mvnrepository.com/artifact/org.bouncycastle/bcprov-jdk16 -->
        <dependency>
            <groupId>org.bouncycastle</groupId>
            <artifactId>bcprov-jdk16</artifactId>
            <version>1.45</version>
        </dependency>
    </dependencies>
</project>
```

## 4.2.8 MQTT Topics

### 4.2.8.1 Device Twin Update

This topic is used to subscribe to device twin updates. It reflects the differences before and after a device twin update.



## Topic

`$hw/events/device/{device_id}/twin/update/document`

Parameter	Type	Description
device_id	String	End device ID.

## Usage

Use an MQTT client to subscribe to this topic.

## Parameter Description

Parameter	Type	Description
event_id	String	Event ID.
timestamp	Int64	Timestamp when the event occurred.
twin	Object	A set of updated device twins. Each twin is in the key-value pair format. The value contains <b>last</b> (twin information before the update) and <b>last</b> (twin information after the update). The twin information contains the option flag, the twin metadata contains the value type, the expected status contains the expected value and update time, and the actual status contains the actual value and update time.

## Example

When an end device is bound to an edge node, the following message is received:

```
$hw/events/device/{device_id}/twin/update/document
{
  "event_id": "",
  "timestamp": 1557314742122,
  "twin": {
    "state": {
      "last": null,
      "current": {
        "expected": {
          "value": "running",
          "metadata": {
            "timestamp": 1557314742122
          }
        }
      },
      "optional": true,
      "metadata": {
        "type": "string"
      }
    }
  }
}
```

## 4.2.8.2 Device Twin Delta

This topic is used to subscribe to device twin delta events. When a device twin changes, the twin properties whose actual values are different from expected values are returned.

### Topic

`$hw/events/device/{device_id}/twin/update/delta`

Parameter	Type	Description
device_id	String	End device ID.

### Usage

Use an MQTT client to subscribe to this topic.

### Parameter Description

Parameter	Type	Description
event_id	String	Event ID.
timestamp	Int64	Timestamp when the event occurred.
twin	Object	A set of updated device twins. Each twin is in the key-value pair format. The value contains the option flag, the twin metadata contains the value type, the expected status contains the expected value and update time, and the actual status contains the actual value and update time.
delta	Map	Names of twin properties whose actual values are different from expected values, and expected values of these properties.

### Example

When an end device is bound to an edge node, the following message is received:

```
$hw/events/device/{device_id}/twin/update/delta
{
  "event_id":"b9625811-f34f-4252-bee9-98185e7e1ec7",
  "timestamp":1557314742131,
  "twin":{
    "state":{
      "expected":{
        "value":"running",
        "metadata":{
          "timestamp":1557314742122
        }
      }
    }
  }
}
```

```

    },
    "optional":true,
    "metadata":{
      "type":"string"
    }
  },
  "delta":{
    "state":"running"
  }
}

```

### 4.2.8.3 Device Member Update

This topic is used to subscribe to updates on bindings between end devices and edge nodes.

#### Topic

\$hw/events/node/{node\_id}/membership/updated

Parameter	Type	Description
node_id	String	Node ID.

#### Usage

Use an MQTT client to subscribe to this topic.

#### Parameter Description

Parameter	Type	Description
event_id	String	Event ID.
timestamp	Int64	Timestamp when the event occurred.
added_devices	Array	Information about end devices, including the device ID, name, properties, and twin properties.
removed_devices	Array	Information about end devices, including the device ID, name, properties, and twin properties.

#### Example

When an end device is bound to an edge node, the following message is received:

```

$hw/events/node/{node_id}/membership/updated
{
  "event_id":"04a975ab-fd51-49be-85f5-5967e994f640",
  "timestamp":1557314742136,
  "added_devices":[
    {
      "id":"ab39361a-6fc0-4c94-b919-72b1e08ca690",
      "name":"IEF-device",

```

```

"state":"unknown",
"attributes":{
  "address":{
    "value":"shenzhen",
    "optional":true,
    "metadata":{
      "type":"string"
    }
  }
},
"twin":{
  "state":{
    "expected":{
      "value":"running",
      "metadata":{
        "timestamp":1557314434570
      }
    },
    "optional":true,
    "metadata":{
      "type":"string"
    }
  }
},
"removed_devices":null
}

```

#### 4.2.8.4 Device Property Update

This topic is used to subscribe to updates on end device properties.

#### Topic

\$hw/events/device/{device\_id}/updated

Parameter	Type	Description
device_id	String	End device ID.

#### Usage

Use an MQTT client to subscribe to this topic.

#### Parameter Description

Parameter	Type	Description
event_id	String	Event ID.
timestamp	Int64	Timestamp when the event occurred.
attributes	Object	A set of updated end device properties, which are in the key-value pair format. The key indicates the property name, the value contains the property value and the optional flag, and metadata contains the value type.

## Example

When an end device is bound to an edge node, the following message is received:

```
$hw/events/device/{device_id}/updated
{
  "event_id": "",
  "timestamp": 1557314742136,
  "attributes": {
    "address": {
      "value": "shenzhen",
      "optional": true,
      "metadata": {
        "type": "string"
      }
    }
  }
}
```

### 4.2.8.5 Device Member Acquisition

This topic is used to publish the request for obtaining end device member information.

## Topic

\$hw/events/node/{node\_id}/membership/get

Parameter	Type	Description
node_id	String	Node ID.

## Usage

Use an MQTT client to publish this topic. This topic must be used together with [Device Member Acquisition Result](#).

## Parameter Description

Parameter	Type	Description
event_id	String	Event ID, which is user-defined.

## Example

```
$hw/events/node/3fbb5b8d-32db-4271-a34f-a013e021b6ce/membership/get
{
  "event_id": "bc876bc-345d-4050-86a8-319a5b13cc10"
}
```

### 4.2.8.6 Device Member Acquisition Result

This topic is used to subscribe to the result of obtaining end device member information.

## Topic

`$hw/events/node/{node_id}/membership/get/result`

Parameter	Type	Description
node_id	String	Node ID.

## Usage

Use an MQTT client to subscribe to this topic. This topic must be used together with [Device Member Acquisition](#).

## Parameter Description

Parameter	Type	Description
event_id	String	Event ID.
timestamp	Int64	Timestamp when the event occurred.
devices	Array	Information about end devices, including the device ID, name, and properties. The properties are in the key-value pair format. The key indicates the property name, the value contains the property value and the optional flag, and metadata contains the value type.

## Example

```
$hw/events/node/3fbb5b8d-32db-4271-a34f-a013e021b6ce/membership/get/result
{
  "event_id":"bc876bc-345d-4050-86a8-319a5b13cc10",
  "timestamp":1557317193524,
  "devices":[
    {
      "id":"ab39361a-6fc0-4c94-b919-72b1e08ca690",
      "name":"IEF-device",
      "state":"unknown",
      "attributes":{
        "address":{
          "value":"longgang",
          "optional":true,
          "metadata":{
            "type":"string"
          }
        }
      }
    }
  ]
}
```

### 4.2.8.7 Device Twin Acquisition

This topic is used to publish the request for obtaining device twins.

## Topic

\$hw/events/device/{device\_id}/twin/get

Parameter	Type	Description
device_id	String	End device ID.

## Usage

Use an MQTT client to publish this topic. This topic must be used together with [Device Twin Acquisition Result](#).

## Parameter Description

Parameter	Type	Description
event_id	String	Event ID.

## Example

```
$hw/events/device/ab39361a-6fc0-4c94-b919-72b1e08ca690/twin/get
{
  "event_id":"123456"
}
```

### 4.2.8.8 Device Twin Acquisition Result

This topic is used to subscribe to the result of obtaining device twins.

## Topic

\$hw/events/device/{device\_id}/twin/get/result

Parameter	Type	Description
device_id	String	Device ID.

## Usage

Use an MQTT client to subscribe to this topic. This topic must be used together with [Device Twin Acquisition](#).

## Parameter Description

Parameter	Type	Description
event_id	String	Event ID.

Parameter	Type	Description
timestamp	Int64	Timestamp when the event occurred.
twin	Object	A set of end device twin information. Each twin is in the key-value pair format. The value contains the option flag, the twin metadata contains the value type, the expected status contains the expected value and update time, and the actual status contains the actual value and update time.

## Example

```
$hw/events/device/ab39361a-6fc0-4c94-b919-72b1e08ca690/twin/get/result
{
  "event_id":"123456",
  "timestamp":1557317510926,
  "twin":{
    "state":{
      "expected":{
        "value":"stop",
        "metadata":{
          "timestamp":1557316778931
        }
      },
      "optional":true,
      "metadata":{
        "type":"string"
      }
    }
  }
}
```

### 4.2.8.9 Device Twin Modification

This topic is used to publish device twin modifications.

## Topic

```
$hw/events/device/{device_id}/twin/update
```

Parameter	Type	Description
device_id	String	End device ID.

## Usage

Use an MQTT client to publish this topic. This topic must be used together with [Device Twin Modification Result](#).



## Parameter Description

Parameter	Type	Description
event_id	String	Event ID.
timestamp	Int64	Timestamp when the event occurred.
twin	Object	A set of device twin information to be modified. The twin property is in the key-value pair format. The key is the name of the twin property to be modified, and the value contains the expected value of the expected status to be modified or the actual value of the actual status to be modified.

## Example

```
$hw/events/device/ab39361a-6fc0-4c94-b919-72b1e08ca690/twin/update
{
  "event_id":"123457",
  "twin":{
    "state":{
      "actual":{
        "value":"stop"
      }
    }
  }
}
```

### 4.2.8.10 Device Twin Modification Result

This topic is used to subscribe to the result of modifying device twins.

## Topic

\$hw/events/device/{device\_id}/twin/update/result

Parameter	Type	Description
device_id	String	End device ID.

## Usage

Use an MQTT client to subscribe to this topic. This topic must be used together with [Device Twin Modification](#).

## Parameter Description

Parameter	Type	Description
event_id	String	Event ID.

Parameter	Type	Description
timestamp	Int64	Timestamp when the event occurred.
twin	Object	A set of modified device twins. Each twin is in the key-value pair format. The value contains the option flag, the twin metadata contains the value type, the expected status contains the expected value and update time, and the actual status contains the actual value and update time.

## Example

```
$hw/events/device/ab39361a-6fc0-4c94-b919-72b1e08ca690/twin/update/result
```

```
{
  "event_id": "123457",
  "timestamp": 1557317614026,
  "twin": {
    "state": {
      "actual": {
        "value": "stop",
        "metadata": {
          "timestamp": 1557317614026
        }
      }
    },
    "optional": true,
    "metadata": {
      "type": "string"
    }
  }
}
```

### 4.2.8.11 Encryption Data Request

This topic is used to publish the request for obtaining encryption data.

## Topic

```
$hw/{project_id}/encryptdatas/{encryptdata_name}/properties/{properties_name}/  
decrypt
```

Parameter	Type	Description
project_id	String	Project ID. For details on how to obtain a project ID, see <a href="#">Obtaining a Project ID</a> .
encryptdata_name	String	Name of an encryption data record.
properties_name	String	Key of an encryption item.

## Usage

Use an MQTT client to publish this topic. This topic must be used together with [Encryption Data Acquisition](#).

A certificate must be used for security authentication when encryption data is requested. For details about the authentication method, see [Performing Security Authentication Using Certificate](#).

## Parameter Description

None.

## Example

Publish an empty message.

### 4.2.8.12 Encryption Data Acquisition

This topic is used to subscribe to encryption data.

## Topic

`$hw/{project_id}/encryptdatas/{encryptdata_name}/properties/{properties_name}/plaintext`

Parameter	Type	Description
project_id	String	Project ID. For details on how to obtain a project ID, see <a href="#">Obtaining a Project ID</a> .
encryptdata_name	String	Name of an encryption data record.
properties_name	String	Key of an encryption item.

## Usage

Use an MQTT client to subscribe to this topic. This topic must be used together with [Encryption Data Request](#).

A certificate must be used for security authentication when encryption data is requested. For details about the authentication method, see [Performing Security Authentication Using Certificate](#).

## Parameter Description

Parameter	Type	Description
plain_text	String	Plaintext value of an encryption item.

Parameter	Type	Description
ret_message	String	Error message.

## Example

If the request is correct, the following message is received:

```
{
  "ret_code": 200,
  "plain_text": "xxxxxxxxxx"
}
```

If the request is incorrect, the following message is received:

```
{
  "ret_code": 400,
  "ret_message": "xxxxxxxxxx"
}
```

### 4.2.8.13 Alarm Reporting

This topic is used to report alarms to AOM.

## Topic

\$hw/alarm/{appname}/add

Parameter	Type	Description
appname	String	Application name, which is user-defined.

## Usage

Use an MQTT client to publish this topic.

## Parameter Description

Parameter	Type	Description
alarmName	String	Alarm name.
alarmId	String	Alarm ID, which must be unique. For details, see <a href="#">Generating an alarmId</a> .
detailedInformation	String	Alarm description.
url	String	URL for root cause analysis. If the URL is not available, leave this field blank.

Parameter	Type	Description
source	String	Alarm source. The value is a character string consisting of uppercase and lowercase letters.
cleared	Boolean	Whether the alarm is cleared. <ul style="list-style-type: none"> <li>● <b>true</b>: The alarm has been cleared.</li> <li>● <b>false</b>: The alarm needs to be cleared manually.</li> </ul>
policyID	String	ID of an alarm rule. For a threshold rule, set this parameter to <b>ruleId</b> . If no threshold rule is configured, leave this field blank.
objectInstance	String	Location information. If this field is not specified, the value of this parameter is the same as that of <b>alarmId</b> by default.
perceivedSeverity	Integer	Alarm severity. <ul style="list-style-type: none"> <li>● 1: Critical</li> <li>● 2: Major</li> <li>● 3: Minor</li> <li>● 4: Warning</li> </ul>
resourceId	Object	Alarm information. For details, see <a href="#">Table 4-6</a> .
resourceType	String	Type of the resource for which the alarm is generated. <ul style="list-style-type: none"> <li>● Application</li> <li>● DB</li> <li>● Host</li> </ul>
eventType	Integer	Alarm type. <ul style="list-style-type: none"> <li>● 21: dynamic threshold alarm</li> <li>● 22: batch threshold alarm</li> <li>● 23: threshold alarm</li> <li>● 24: system alarm</li> <li>● 25: probe added or deleted</li> <li>● 26: Agent installation alarm</li> <li>● 27: Quota threshold-crossing alarm</li> </ul>
probableCause	String	Possible causes.
proposedRepairActions	String	Handling suggestions.

**Table 4-6** resourceId

Parameter	Type	Description
namespace	String	Resource type. The options are as follows: <ul style="list-style-type: none"> <li>● <b>PAAS.CONTAINER</b>: container metrics</li> <li>● <b>PAAS.NODE</b>: node metrics</li> </ul>
dimension	Object	Dimension information, which is associated with the node application information reported by the monitoring module. For details, see <a href="#">Table 4-7</a> . Currently, alarms are about nodes and applications. Therefore, you only need to pay attention to these two dimensions. The application dimension covers service, instance, container, and process information. You can select one or more types of information to be reported. <ul style="list-style-type: none"> <li>● Service information: clusterId, nameSpace, and serviceID</li> <li>● Instance information: podID and podName</li> <li>● Container information: containerID and containerName</li> <li>● Process information: processID and processName</li> </ul>

**Table 4-7** Dimension

Parameter	Type	Description
clusterId	String	Project ID. For details on how to obtain a project ID, see <a href="#">Obtaining a Project ID</a> .
nameSpace	String	The default value is <b>default</b> .
nodeIP	String	Node IP address.

Parameter	Type	Description
serviceID	String	<p>Service ID.</p> <ul style="list-style-type: none"> <li>For applications, the value is the MD5 value of {projectId}_{hostid}_{appName}.</li> <li>For processes, the value is the MD5 value of {projectId}_{hostid}_{Process name}_{Process pid}.</li> <li><b>processID</b> can be calculated as follows: md5({projectId})_{hostid}_md5({Process name})_{process pid}. <b>md5(projectId)</b> indicates the MD5 value of <b>projectId</b>, and <b>md5(Process name)</b> indicates the MD5 value of the process name.</li> </ul> <p>The dimension settings must be consistent with those reported by the monitoring module. Otherwise, the corresponding resources cannot be associated.</p>
podID	String	Instance ID.
podName	String	Instance name.
containerID	String	Container ID.
containerName	String	Container name.
processID	String	Process ID.
processName	String	Process name.
Application	String	Application name.

## Generating an alarmId

### NOTE

You do not need to use this method to generate the alarmId. However, you need to ensure that the alarmId is unique.

Generate the MD5 value of **{projectId}\_{Service name for which the alarm is generated}\_{Dimension information}\_{Metric name}\_{Alarm type}\_{Rule information}**.

Where:

- **projectId**: Project ID. For details on how to obtain a project ID, see [Obtaining a Project ID](#).
- **Dimension information**:
  - Node information: {clusterId}\_{namespace}\_{ip}

- Container information: {clusterId}\_{namespace}\_{appName}\_{podName}\_{containerId}
- Application information: {clusterId}\_{namespace}\_{appName}
- **Alarm type:**
  - 21: dynamic threshold alarm
  - 22: batch threshold alarm
  - 23: threshold alarm
  - 24: system alarm
  - 25: probe added or deleted
  - 26: Agent installation alarm
- **Rule information:** Set this field to a rule name for threshold alarms and to **NA** for alarms generated by the service itself. For dynamic threshold alarms, set this field to **policyId**.

## Example

- Node alarm

```
{
  "alarmName": "test",
  "alarmId": "73ccbccce05de74f9d3dda42f6ecfe20",
  "detailedInformation": "test",
  "url": "",
  "source": "IEF",
  "cleared": false,
  "policyID": "",
  "perceivedSeverity": 4,
  "resourceId": {
    "namespace": "PAAS.NODE",
    "dimension": {
      "clusterId": "e277befa37a64ed1aa25b522e686bc28",
      "nameSpace": "default",
      "nodeIP": "192.168.0.164"
    }
  }
},
"neType": "Host",
"eventType": 23
}
```

- Application alarm:

```
{
  "alarmName": "Application restart",
  "alarmId": "b09076ff565c59d4da0db0c9223781",
  "detailedInformation": "Application restart test",
  "url": "",
  "source": "IEF",
  "cleared": false,
  "policyID": "",
  "perceivedSeverity": 3,
  "resourceId": {
    "namespace": "PAAS.CONTAINER",
    "dimension": {
      "containerName": "container-e991acd3-864c-4038-8a90-e042eebab496",
      "containerID": "70b385315c8ac507b3de7dfe1258932cea0b53a850b7d030ce7ed0a55c47877c",
      "podID": "0e9ce4fd-b732-11e9-8a30-fa163e9b3546",
      "podName": "hxxkapp1-7898f5bd4b-2lj8z"
    }
  }
},
"neType": "Application",
"eventType": 23
}
```



#### 4.2.8.14 Alarm Clearance

This topic is used to clear alarms in AOM. The message body is the same as that described in [Alarm Reporting](#). To be specific, the only difference between the requests for reporting and clearing alarms is the topic name.

#### Topic

\$hw/alarm/{appname}/clear

Parameter	Type	Description
appname	String	Application name, which is user-defined.

#### Usage

Use an MQTT client to publish this topic.

#### Parameter Description

See [Alarm Reporting](#).

#### 4.2.8.15 Custom Topics

IEF allows you to customize topics based on actual requirements.

To use a custom topic, create a message route in which a topic is defined on the IEF console. IEF will forward the data in the topic based on the message route. For details on how to create a message route, see [Edge-Cloud Message Overview](#).

#### Topic

{project\_id}/nodes/{node\_id}/user/{custom\_topic}

#### Usage

Use an MQTT client to publish this topic.

#### Parameter Description

IEF can transparently forward any content in a custom topic.

**{custom\_topic}** supports the wildcards **#** and **+**. Therefore, multiple messages that meet a wildcard rule can be forwarded in a unified manner.

**#** is a wildcard that matches any number (0 or greater) of levels in a topic. **+** is a wildcard that matches only one level in a topic.

IEF performs the minimum match on the topic messages that match the wildcard rule and then forwards the messages. For example, you can configure 123/+/567 as a wildcard rule for message topics 123/aaa/567 and 123/bbb/567.

## Example

After a route is created, you can publish messages (data) to the custom topic defined in the route. IEF then forwards the messages to the specified endpoint. IEF also records the number of forwarded messages, including successful and failed messages.

**Figure 4-29** Number of forwarded messages

Route Na...	Source Endpoint	Destination Endpoint	Status	Forwarded Messages
<input type="checkbox"/> asaadaaaa	SystemREST <span>Cloud</span>	service-8090 <span>Edge</span>	<span>Enabled</span>	Total: 0 Successful: 0 Failed: 0
<input type="checkbox"/> --	/ada	/asdaaa		

## 4.3 Containerized Application Management

### 4.3.1 Containerized Applications

IEF delivers containerized applications to edge nodes. This section describes how to create a custom edge application.

#### Constraints

- When the disk usage of an edge node exceeds 70%, the image reclamation mechanism is started to reclaim the disk space occupied by the container image. In this case, if you deploy a containerized application, the application container will take a long time to start. Therefore, properly plan the disk space of the edge node before deploying the containerized application.
- When you create a containerized application, the edge node pulls a container image from SWR. If the image is too large and the edge node has a limited download bandwidth, the image fails to be pulled. In this case, a message will be displayed on the IEF console, indicating that the containerized application fails to be created. However, the image pull will not be interrupted. After the container image is pulled successfully, the containerized application will be automatically created. To prevent such a problem, you can pull the container image to the edge node and then create a containerized application.
- The architecture of the container image to be used must be the same as that of the edge node on which a containerized application is to be deployed. For example, if the edge node uses x86, the container image must also use x86.
- In a platinum service instance, the number of application instances can be changed to zero.

### Creating an Edge Application

- Step 1** Log in to the IEF console, and click **Switch Instance** on the **Dashboard** page to select a platinum service instance.
- Step 2** In the navigation pane, choose **Edge Applications > Containerized Applications**. Then, click **Create Containerized Application** in the upper right corner.
- Step 3** Specify basic information.

- **Name:** name of a containerized application.
- **Instances:** number of instances of a containerized application. Multiple instances are allowed.
- **Configuration Method**
  - **Custom:** Configure the application step by step. For details, see [Step 4 to Step 6](#).
  - **Application template:** Select a predefined application template and modify it based on your requirements. This method reduces repeated operations. The configurations in a template are the same as those set in [Step 4 to Step 6](#). For details on how to create a template, see [Application Templates](#).
- **Tags**  
Tags can be used to classify resources, facilitating resource management.

**Figure 4-30 Basic information**

Creation Mode	Custom
* Name	<input type="text" value="Enter an application name."/>
Instances	1
Configuration Method	<input checked="" type="radio"/> Custom <input type="radio"/> Application template
Description	<input type="text" value="Enter a description."/> <div style="text-align: right;">0/255</div>
Tags	<input type="text" value="Tag key"/> <input type="text" value="Tag value"/>

You can add 20 more tags.

**Step 4** Configure containers.

Click **Use Image** under the image to be used to deploy the application.

- **My Images:** shows all the images you have created in [SWR](#).
- **Shared Images:** displays images shared by other users. Shared images are managed and maintained in SWR. For details, see [Sharing a Private Image](#).

After selecting an image, specify container configurations.

- **Image Version:** Select the version of the image to be used to deploy the application.

**NOTICE**

Do not use version **latest** when deploying containers in the production environment. Otherwise, it will be difficult to determine the version of the running image and to roll back the application properly.

- **Container Specifications:** Specify CPU, memory, Ascend AI accelerator card, and GPU quotas.
- **Ascend AI accelerator card:** The AI accelerator card configuration of the containerized application must be the same as that of the edge node actually deployed. Otherwise, the application will fail to be created. For details, see [Registering an Edge Node](#).

 **NOTE**

For NPUs after virtualization partition, only one virtualized NPU can be mounted to a container. The virtualized NPU can be allocated to another container only after the original container quits.

The following table lists the NPU types supported by AI accelerator cards.

**Table 4-8** NPU types

Type	Description
Ascend 310	Ascend 310 chips
Ascend 310B	Ascend 310B chips
Ascend 910	Ascend 910 chips
Ascend 310P	Ascend 310P chips
Ascend 310P-share	Ascend 310P chips that support multi-container sharing
Ascend 310P-1c	Ascend 310P chips after virtualization partition based on the template <b>vir01</b>
Ascend 310P-2c	Ascend 310P chips after virtualization partition based on the template <b>vir02</b>
Ascend 310P-2c.1cpu	Ascend 310P chips after virtualization partition based on the template <b>vir02_1c</b>
Ascend 310P-4c	Ascend 310P chips after virtualization partition based on the template <b>vir04</b>
Ascend 310P-4c.3cpu	Ascend 310P chips after virtualization partition based on the template <b>vir04_3c</b>

**Figure 4-31** Container configuration

The screenshot shows a configuration form for a container. At the top, the 'Image Name' is 'nginx' with a 'Change Image' link. Below it, 'Image Version' is set to 'latest' and 'Container Name' is 'container-75'. The 'Container Specifications' section includes:
 

- CPU:** Request and Limit are both set to 0.25 cores.
- Memory:** Request and Limit are both set to 512.00 MiB.
- AI Accelerator Card:** A dropdown menu shows 'Not installed' as the selected option, with other options being 'Ascend AI accelerator card' and 'NVIDIA GPU'.

You can also configure the following advanced settings for the container:

- **Command**

A container image has metadata that stores image details. If lifecycle commands and arguments are not set, IEF runs the default commands and arguments provided during image creation, that is, **ENTRYPOINT** and **CMD** in the Dockerfile.

If the commands and arguments used to run a container are set during application creation, the default commands **ENTRYPOINT** and **CMD** are overwritten during image building. The rules are as follows:

**Table 4-9** Commands and arguments used to run a container

Image ENTRYPOINT	Image CMD	Command to Run a Container	Arguments to Run a Container	Command Executed
[touch]	[/root/test]	Not set	Not set	[touch /root/test]
[touch]	[/root/test]	[mkdir]	Not set	[mkdir]
[touch]	[/root/test]	Not set	[/opt/test]	[touch /opt/test]
[touch]	[/root/test]	[mkdir]	[/opt/test]	[mkdir /opt/test]

**Figure 4-32** Command

The screenshot shows the 'Advanced Settings' for a container's command configuration. It includes a warning: 'The information entered here will be viewable. Do not enter sensitive information.' The 'Command' field contains '/run/start' and the 'Arguments' field contains '--port=8080'. A 'Demo' section on the right shows a preview of the configuration with 'Binary' selected and 'Bash' as an alternative. A page number '11/65,535' is visible at the bottom right.

- **Command**

Enter an executable command, for example, **/run/start**.

If there are multiple commands, separate them with spaces. If the command contains a space, enclose the command in quotation marks ("").

 **NOTE**

In the case of multiple commands, you are advised to run **/bin/sh** or other **shell** commands. Other commands are used as arguments.

- **Arguments**

Enter the argument that controls the container running command, for example, **--port=8080**.

If there are multiple arguments, separate them with line breaks.

- **Security Options**

- You can enable **Privileged Mode** to grant root permissions to the container for accessing host devices (such as GPUs and FPGAs).

- **RunAsUser Switch**

By default, IEF runs the container as the user defined during image building.

You can specify a user to run the container by turning this switch on, and entering the user ID (an integer ranging from 0 to 65534) in the text box displayed. If the OS of the image does not contain the specified user ID, the application fails to be started.

- **Environment Variables**

An environment variable affects the way a running container will behave. Variables can be modified after workload deployment. Currently, environment variables can be manually added, imported from secrets or ConfigMaps, or referenced from **hostIP**.

- **Added manually:** Customize a variable name and value.
- **Added from Secret:** You can customize a variable name. The variable value is referenced from secret configuration data. For details on how to create a secret, see [Secrets](#).
- **Added from ConfigMap:** You can customize a variable name. The variable value is referenced from ConfigMap configuration data. For details on how to create a ConfigMap, see [ConfigMaps](#).
- **Variable reference:** The variable value is referenced from **hostIP**, that is, the IP address of an edge node.

 **NOTE**

IEF does not encrypt the environment variables you entered. If the environment variables you attempt to configure contain sensitive information, you need to encrypt them before entering them and also need to decrypt them when using them.

IEF does not provide any encryption and decryption tools. If you need to configure cypher text, choose your own encryption and decryption tools.

- **Data Storage**

You can define a local volume and mount the local storage directory of the edge node to the container for persistent data storage.

Currently, the following four types of local volumes are supported:

- **hostPath**: used for mounting a host directory to the container. hostPath is a persistent volume. After an application is deleted, the data in hostPath still exists in the local disk directory of the edge node. If the application is re-created later, previously written data can still be read after the directory is mounted.

You can mount the application log directory to the **var/IEF/app/log/{appName}** directory of the host. In the directory name, **{appName}** indicates the application name. The edge node will upload the .log and .trace files in the **/var/IEF/app/log/{appName}** directory to AOM.

The mount directory is the log path of the application in the container. For example, the default log path of the Nginx application is **/var/log/nginx**. The permission must be set to **Read/Write**.

**Figure 4-33** Log volume mounting

Local Volume Name	Type	Mount Directory	Permission	Operation
log	hostPath	/var/IEF/app/log/nginx	/var/log/nginx	Read/Write Delete

- **emptyDir**: a simple empty directory used for storing transient data. It can be created in hard disks or memory. The application can read files from and write files into the directory. emptyDir has the same lifecycle as the application. If the application is deleted, the data in emptyDir is deleted along with it.
- **configMap**: a type of resources that store configuration details required by the application. For details on how to create a ConfigMap, see [ConfigMaps](#).
- **secret**: a type of resources that store sensitive data, such as authentication, certificate, and key details. For details on how to create a secret, see [Secrets](#).

#### NOTICE

- The container path cannot be a system directory, such as **/** or **/var/run**. Otherwise, an exception occurs. You are advised to mount the container to an empty directory. If the directory is not empty, ensure that the directory does not contain any files that affect container startup. Otherwise, the files will be replaced, making it impossible for the container to be properly started. As a result, the application creation will fail.
- If the container is mounted into a high-risk directory, you are advised to use an account with minimum permissions to start the container. Otherwise, high-risk files on the host machine may be damaged.

- **Health Check**

Health check regularly checks the status of containers or workloads.

- **Liveness Probe**: The system executes the probe to check if a container is still alive, and restarts the instance if the probe fails. Currently, the system probes a container by HTTP request or command and determines whether the container is alive based on the response from the container.

- **Readiness Probe:** The system invokes the probe to determine whether the instance is ready. If the instance is not ready, the system does not forward requests to the instance.

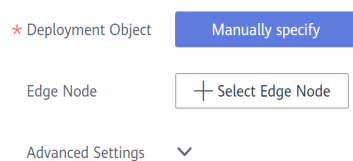
For details, see [Health Check Configuration](#).

**Step 5 Click Next.**

Select a deployment object. Currently, two deployment objects are supported.

- **Manually specify**

**Figure 4-34** Specifying an edge node



- **Automatically schedule**

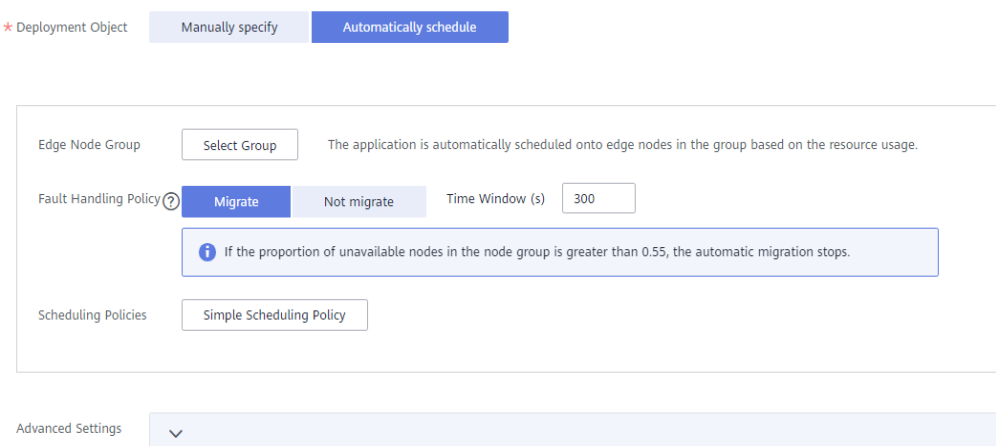
The containerized application will be automatically scheduled in the edge node group based on the resource usage of the nodes. You can also set a restart policy to determine whether the application instance is re-scheduled onto other available nodes in the edge node group when the edge node where the application instance originally runs is unavailable.

**NOTE**

If more than 55% of nodes is unavailable in a node group, automatic migration will stop.

You can also configure scheduling policies. For details, see [Affinity and Anti-Affinity Scheduling](#).

**Figure 4-35** Automatic scheduling



**Time Window:** indicates how long the node will stay bound to the node with the specified taint. If this parameter is not configured, the default value is **300**. During the fault time window, nodes can still be scheduled.

You can also configure the following advanced settings for the container:



- **Restart Policy**
  - **Always restart:** The system restarts the container regardless of whether it had quit normally or unexpectedly.  
This policy is used for a node group.
  - **Restart upon failure:** The system restarts the container only if it had previously quit unexpectedly.
  - **Do not restart:** The system does not restart the container regardless of whether it had quit normally or unexpectedly.

---

**NOTICE**

You are allowed to upgrade a containerized application and modify its instance quantity and access configuration only after you select **Always restart**.

- **Host PID**

Enabling this function allows containers to share the PID namespace with the host where the edge node resides. In this way, you can perform container operations on the edge node, for example, starting and stopping a container. Similarly, you can perform operations related to the edge node in containers, for example, starting and stopping a process of the edge node.

This function can only be enabled on edge nodes running software v2.8.0 or later.

**Step 6** Click **Next**.

Container access supports bridged network and host network.

---

**CAUTION**

If the ports of containers deployed on the same edge node conflict, the containers will fail to be started.

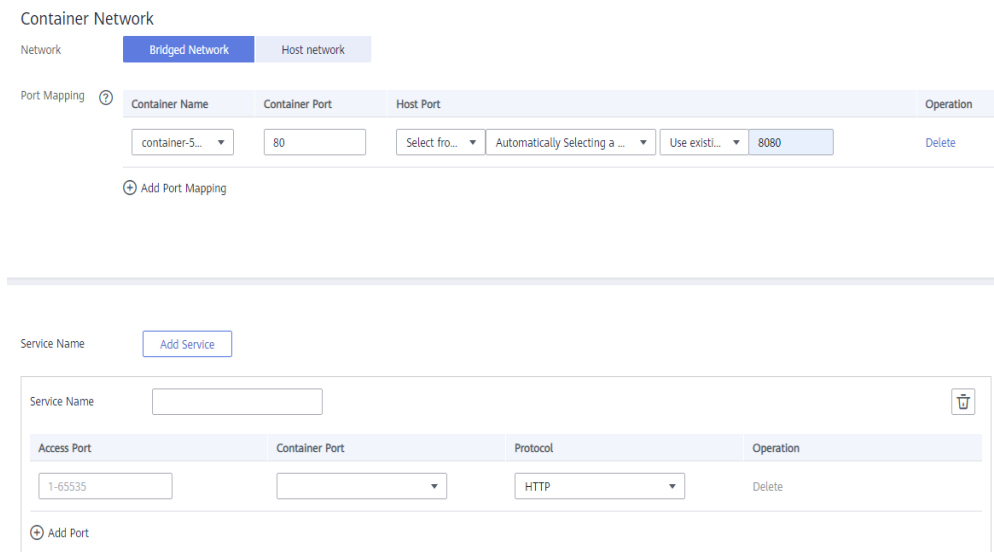
- **Bridged network**
  - The container uses an independent virtual network. Configure relationships between container and host ports to enable external communications. After port mapping is configured, traffic destined for a host port is distributed to the mapping container port. For example, if container port 80 is mapped to host port 8080, the traffic destined for host port 8080 will be directed to container port 80.
  - You can select a host NIC to enable the port bound to this NIC to communicate with the container port. Note that port mapping does not support NICs with IPv6 addresses.
  - The host port in the port mapping can be specified or automatically assigned. Automatically assigning the host port prevents container startup failures caused by port conflicts of multiple instances.

**NOTE**

For automatic port assigning, enter a proper port range to avoid port conflicts.

- Host network**  
 The network of the host (edge node) is used. To be specific, the container and the host use the same IP address, and network isolation is not required between them.
- Service**  
 Click **Add Service**, enter a Service name and access port, and set **Container Port** to the one set in port mapping. The protocol can be **HTTP** or **TCP**.

**Figure 4-36** Adding a Service



If **Bridged network** is selected for **Network**, **Add Service** is not available before you specify ports in **Port Mapping**.

A Service defines more refined access mode for applications. You can create a Service after an application is created. For details, see [Creating a Service](#).

**Step 7** Click **Next**. Confirm the specifications of the containerized application and click **Create**.

----End

## Querying Application O&M Information

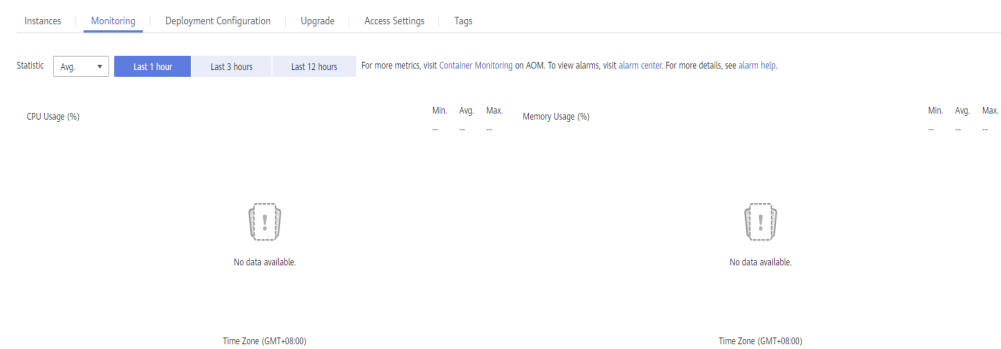
After an application is deployed, you can view the CPU and memory information of the application on the IEF console. You can also go to the **Container Monitoring** page on the AOM console to view additional metrics, and add or view alarms in **Alarm Center**.

**Step 1** Log in to the IEF console, and click **Switch Instance** on the **Dashboard** page to select a platinum service instance.

**Step 2** In the navigation pane, choose **Edge Applications > Containerized Applications**. Then, click an application name.

**Step 3** Click the **Monitoring** tab to view the monitoring details for the application.

**Figure 4-37** Application monitoring details



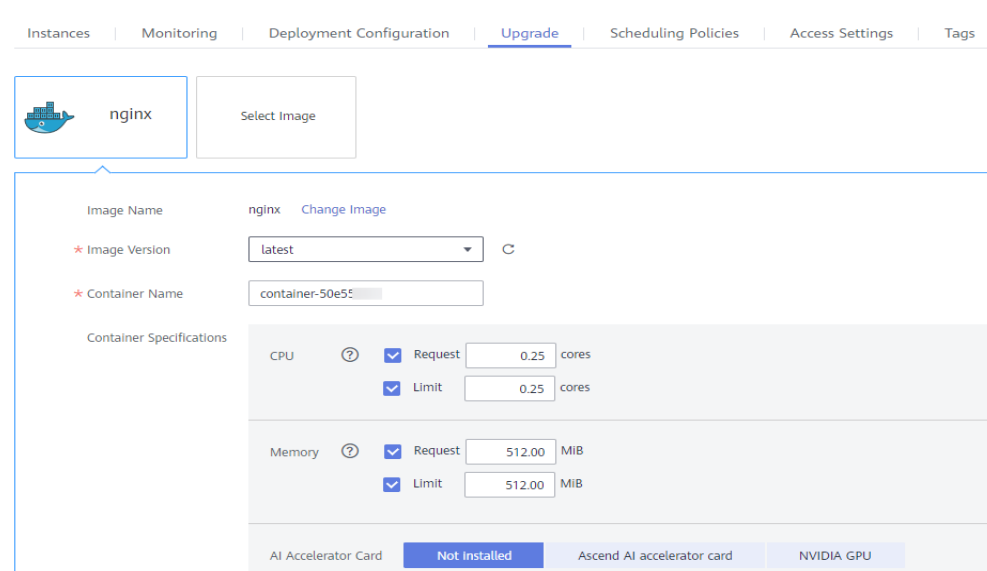
----End

## Upgrading an Application

After an application is deployed, you can upgrade it. Rolling upgrades are used. That is, a new application instance is created first, and then the old application instance is deleted.

- Step 1** Log in to the IEF console, and click **Switch Instance** on the **Dashboard** page to select a platinum service instance.
- Step 2** In the navigation pane, choose **Edge Applications > Containerized Applications**. Then, click an application name.
- Step 3** Click the **Upgrade** tab and modify the container configurations. The parameter settings are the same as those in [Step 4](#).

**Figure 4-38** Upgrading an application



- Step 4** Click **Submit**.

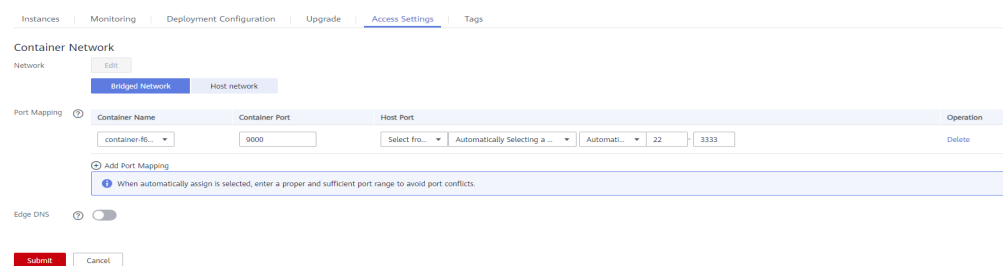
----End

## Modifying Access Settings

After an application is deployed, you can modify the access settings of the application.

- Step 1** Log in to the IEF console, and click **Switch Instance** on the **Dashboard** page to select a platinum service instance.
- Step 2** In the navigation pane, choose **Edge Applications > Containerized Applications**. Then, click an application name.
- Step 3** Click the **Access Settings** tab and modify the configurations. The parameter settings are the same as those in [Step 6](#).

**Figure 4-39** Modifying access settings



- Step 4** Click **Submit**.

-----End

## 4.3.2 Application Templates

An application template defines the details required by containerized applications, such as the container image, configuration, disk mounting information, and resource usage.

Applications are created from an image. You need to create and upload an image to the image repository in advance.

### Creating an Application Template

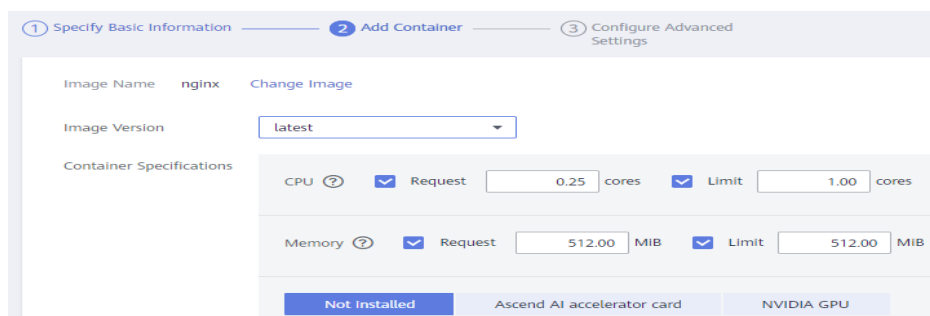
- Step 1** Log in to the IEF console, and click **Switch Instance** on the **Dashboard** page to select a platinum service instance.
- Step 2** In the navigation pane, choose **Edge Applications > Application Templates**. Then, click **Create Application Template** in the upper right corner.
- Step 3** Specify basic information.
  - **Name** (mandatory): name of an application template.
  - **Version** (mandatory): version number of the application template.
  - **Alias**: alias of the application template.
  - **Architecture**: architecture supported by the application.
  - **Description**: description of the application template.
  - **Tags**: Tags can be used to classify application templates, facilitating template management. The tags specified in this step are only used to identify

application templates. You can use these tags to filter the application templates during search.

**Step 4** Click **Next**. Then, add a container.

1. Click **Use Image** under the target image.
  - **My Images**: shows all the images you have uploaded to the image repository.
  - **Shared Images**: shows the images shared by other users.
2. Click **Next**. Then, specify container specifications.

**Figure 4-40** Image information



- **Image Version**: Select the version of the image to be used to deploy the application.
- **Container Specifications**: Specify CPU, memory, Ascend AI accelerator card, and GPU quotas.
- **Ascend AI accelerator card**

The AI accelerator card configuration of the containerized application must be the same as that of the edge node actually deployed. Otherwise, the application will fail to be created. For details, see [Registering an Edge Node](#).

**NOTE**

For NPUs after virtualization partition, only one virtualized NPU can be mounted to a container. The virtualized NPU can be allocated to another container only after the original container quits.

The following table lists the NPU types supported by Ascend AI accelerator cards.

**Table 4-10** NPU types

Type	Description
Ascend 310	Ascend 310 chips
Ascend 310B	Ascend 310B chips
Ascend 910	Ascend 910 chips
Ascend 310P	Ascend 310P chips

Type	Description
Ascend 310P-share	Ascend 310P chips that support multi-container sharing
Ascend 310P-1c	Ascend 310P chips after virtualization partition based on the template <b>vir01</b>
Ascend 310P-2c	Ascend 310P chips after virtualization partition based on the template <b>vir02</b>
Ascend 310P-2c.1cpu	Ascend 310P chips after virtualization partition based on the template <b>vir02_1c</b>
Ascend 310P-4c	Ascend 310P chips after virtualization partition based on the template <b>vir04</b>
Ascend 310P-4c.3cpu	Ascend 310P chips after virtualization partition based on the template <b>vir04_3c</b>

**Step 5** Click **Next**. Then, configure the application.

- **Startup Commands**

A container image has metadata that stores image details. If lifecycle commands and arguments are not set, IEF runs the default commands and arguments provided during image creation, that is, **ENTRYPOINT** and **CMD** in the Dockerfile.

If the commands and arguments used to run a container are set during application creation, the default commands **ENTRYPOINT** and **CMD** are overwritten during image building. The rules are as follows:

**Table 4-11** Commands and arguments used to run a container

Image ENTRYPOINT	Image CMD	Command to Run a Container	Arguments to Run a Container	Command Executed
[touch]	[/root/test]	Not set	Not set	[touch /root/test]
[touch]	[/root/test]	[mkdir]	Not set	[mkdir]
[touch]	[/root/test]	Not set	[/opt/test]	[touch /opt/test]
[touch]	[/root/test]	[mkdir]	[/opt/test]	[mkdir /opt/test]

**Figure 4-41** Startup command

The information entered here will be viewable. Do not enter sensitive information.

Command:

Arguments:

Demo:  Binary  Bash

Command:

Arguments:

#### – Command

Enter an executable command, for example, **/run/start**.

If there are multiple commands, separate them with spaces. If the command contains a space, enclose the command in quotation marks ("").

#### 📖 NOTE

In the case of multiple commands, you are advised to run **/bin/sh** or other **shell** commands. Other commands are used as arguments.

#### – Arguments

Enter the argument that controls the container running command, for example, **--port=8080**.

If there are multiple arguments, separate them with line breaks.

#### ● Environment Variables

An environment variable affects the way a running container will behave. You can modify environment variables even after applications are deployed.

Click **Add Environment Variable** to add a variable. Currently, environment variables can be manually added, imported from secrets or ConfigMaps, or referenced from **hostIP**.

– **Added manually:** Customize a variable name and value.

– **Added from Secret:** You can customize a variable name. The variable value is referenced from secret configuration data. For details on how to create a secret, see [Secrets](#).

– **Added from ConfigMap:** You can customize a variable name. The variable value is referenced from ConfigMap configuration data. For details on how to create a ConfigMap, see [ConfigMaps](#).

– **Variable reference:** The variable value is referenced from **hostIP**, that is, the IP address of an edge node.

#### 📖 NOTE

IEF does not encrypt the environment variables you entered. If the environment variables you attempt to configure contain sensitive information, you need to encrypt them before entering them and also need to decrypt them when using them.

IEF does not provide any encryption and decryption tools. If you need to configure cypher text, choose your own encryption and decryption tools.

#### ● Volumes

A volume is used for storage when the container is running. Currently, the following four volume types are supported:

- **hostPath**: used for mounting a host directory to the container. hostPath is a persistent volume. After an application is deleted, the data in hostPath still exists in the local disk directory of the edge node. If the application is re-created later, previously written data can still be read after the directory is mounted.

You can mount the application log directory to the **var/IEF/app/log/{appName}** directory of the host. In the directory name, **{appName}** indicates the application name. The edge node will upload the .log and .trace files in the **/var/IEF/app/log/{appName}** directory to AOM.

Figure 4-42 Log volume mounting

Local Volume Name	Type	Mount Directory	Permission	Operation
log	hostPath	/var/IEF/app/log/applicat	Read only	Delete

⊕ Add Volume

- **emptyDir**: a simple empty directory used for storing transient data. It can be created in hard disks or memory. emptyDir is an empty directory after being mounted. The application can read files from and write files into the directory. emptyDir has the same lifecycle as the application. If the application is deleted, the data in emptyDir is deleted along with it.
- **configMap**: a type of resources that store configuration details required by the application. For details on how to create a ConfigMap, see [ConfigMaps](#).
- **secret**: a type of resources that store sensitive data, such as authentication, certificate, and key details. For details on how to create a secret, see [Secrets](#).

#### NOTICE

- The container path cannot be a system directory, such as **/** or **/var/run**. Otherwise, an exception occurs. You are advised to mount the container to an empty directory. If the directory is not empty, ensure that the directory does not contain any files that affect container startup. Otherwise, the files will be replaced, making it impossible for the container to be properly started. As a result, the application creation will fail.
- If the container is mounted into a high-risk directory, you are advised to use an account with minimum permissions to start the container. Otherwise, high-risk files on the host machine may be damaged.

#### Options

- **Privileged Mode**

You can enable **Privileged Mode** to grant root permissions to the container for accessing host devices (such as GPUs and FPGAs).

- **RunAsUser Switch**

By default, IEF runs the container as the user defined during image building.

You can specify a user to run the container by turning this switch on, and entering the user ID (an integer ranging from 0 to 65534) in the text box



displayed. If the OS of the image does not contain the specified user ID, the application fails to be started.

- **Restart Policy**

**Always restart:** The system restarts the container regardless of whether it had quit normally or unexpectedly.

**Restart upon failure:** The system restarts the container only if it had previously quit unexpectedly.

**Do not restart:** The system does not restart the container regardless of whether it had quit normally or unexpectedly.

- **Container Network**

**Host network:** The network of the host (edge node) is used. To be specific, the container and the host use the same IP address, and network isolation is not required between them.

**Bridged network:** The container uses an independent virtual network. Configure relationships between container and host ports to enable external communications. After port mapping is configured, traffic destined for a host port is distributed to the mapping container port. For example, if container port 80 is mapped to host port 8080, the traffic destined for host port 8080 will be directed to container port 80.

- **Health Check**

- **Liveness Probe:** The system executes the probe to check if a container is still alive, and restarts the instance if the probe fails. Currently, the system probes a container by HTTP request or command and determines whether the container is alive based on the response from the container.

- **Readiness Probe:** The system invokes the probe to determine whether the instance is ready. If the instance is not ready, the system does not forward requests to the instance.

For details, see [Health Check Configuration](#).

**Step 6** After configuration, click **Create**.

----End

## Creating an Application Template Version

You can create multiple application versions for an application template to facilitate application management.

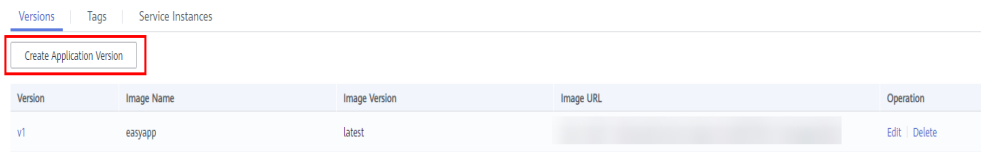
**Step 1** Log in to the IEF console, and click **Switch Instance** on the **Dashboard** page to select a platinum service instance.

**Step 2** In the navigation pane, choose **Edge Applications > Application Templates**.

**Step 3** Click the application template whose version is to be added. The application template details page is displayed.

**Step 4** In the lower left corner of the page, click **Create Application Version**.

**Figure 4-43** Creating an application version



**Step 5** Enter a version, and click **Next**. Then, configure other parameters by following the description in [Creating an Application Template](#).

----End

## Sharing Templates

Application templates can be shared by other platinum service instances in the same region.

### NOTE

Application templates cannot be shared between Platinum Edition v2 or later and earlier versions.

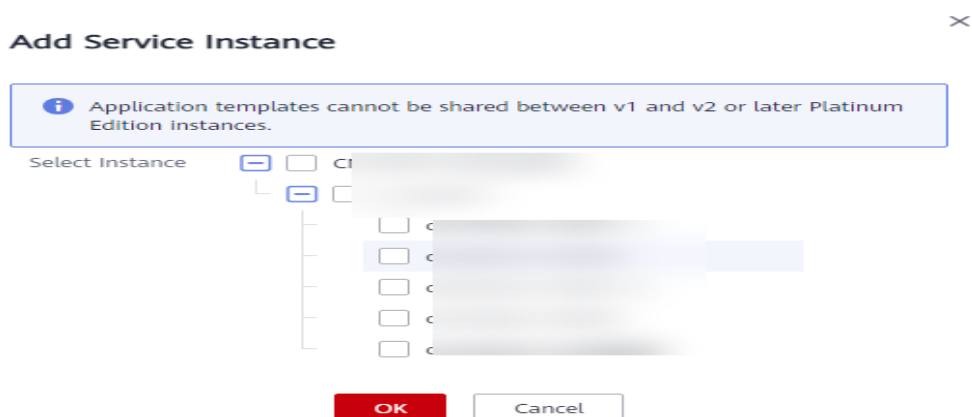
**Step 1** Log in to the IEF console, and click **Switch Instance** on the **Dashboard** page to select a platinum service instance.

**Step 2** In the navigation pane, choose **Edge Applications > Application Templates**.

**Step 3** On the **Private Templates** page, click the application template to be shared. The application template details page is displayed.

**Step 4** Click the **Service Instances** tab, click **Add Service Instance**, and select service instances.

**Step 5** Click **OK**.



----End

## 4.3.3 Affinity and Anti-Affinity Scheduling

### Affinity and Anti-Affinity Scheduling Policies

When creating a containerized application, you can set affinity and anti-affinity scheduling policies. For example, you can deploy applications of a certain type on specific nodes or deploy different applications on different nodes.

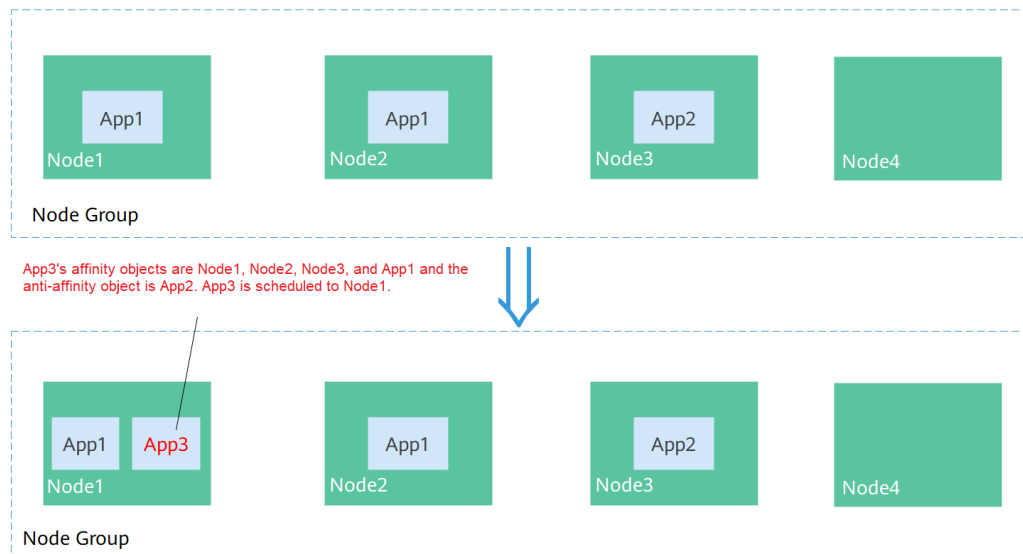
IEF supports the following simple scheduling policies:

- Affinity
  - Application-node affinity: If the affinity is defined between a containerized application and certain nodes, the application will be scheduled only to these nodes. If you have set the affinity between an application and certain nodes, you do not need to set the anti-affinity between the application and other nodes because the affinity determines that the application can be scheduled only to the specified nodes.
  - Affinity between containerized applications: If the affinity is defined between containerized applications A and B, application A will be scheduled only to the nodes where application B is located.
- Anti-Affinity
  - Application-node anti-affinity: If the anti-affinity is defined between a containerized application and certain nodes, the application will not be scheduled to these nodes.
  - Anti-affinity between containerized applications: If the anti-affinity is defined between containerized applications A and B, application A will not be scheduled to the nodes where application B is located.

### Affinity and Anti-Affinity Policy Example

You can configure multiple affinity and anti-affinity objects for an application. For example, a node group contains four nodes: Node1, Node2, Node3, and Node4. Instances of App1 run on Node1 and Node2, and instances of App2 run on Node3. If you create App3 (Node1, Node2, Node3, and App1 set as the affinity objects and App2 set as the anti-affinity object), App3 will be scheduled only to Node1 and Node2. As shown in the following figure, App3 has only one instance, which is scheduled to Node1.

**Figure 4-44** App3 scheduled to Node1

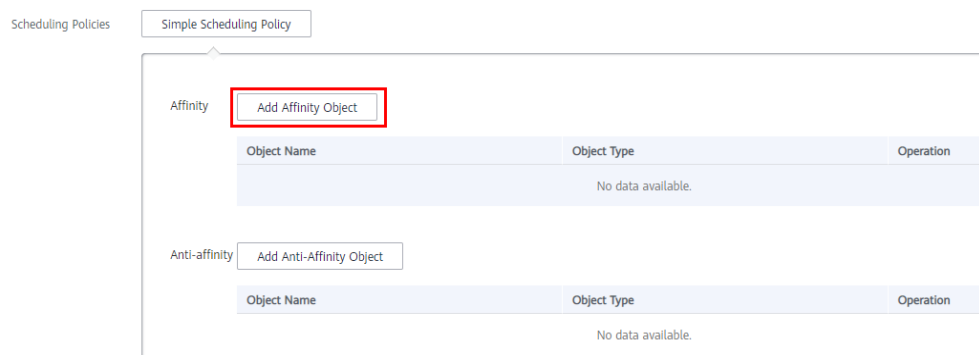


## Adding an Affinity Object

Once you have selected an edge node group in the **Associate Object** step during creation of a containerized application, you can configure an affinity policy.

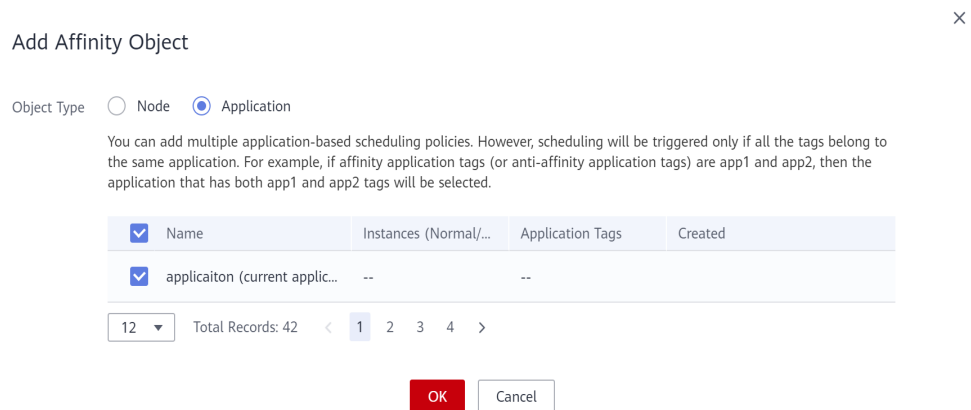
**Step 1** Click **Add Affinity Object**.

**Figure 4-45** Adding an affinity object



**Step 2** In the dialog box displayed, select one or more affinity nodes or applications and click **OK**.

**Figure 4-46** Selecting an affinity object



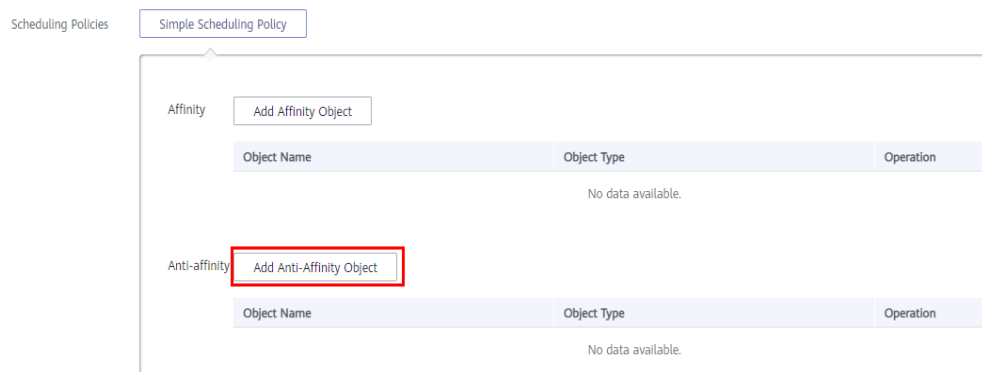
----End

## Adding an Anti-affinity Object

You can configure an anti-affinity policy after you select an edge node group in the **Associate Object** step during creation of a containerized application.

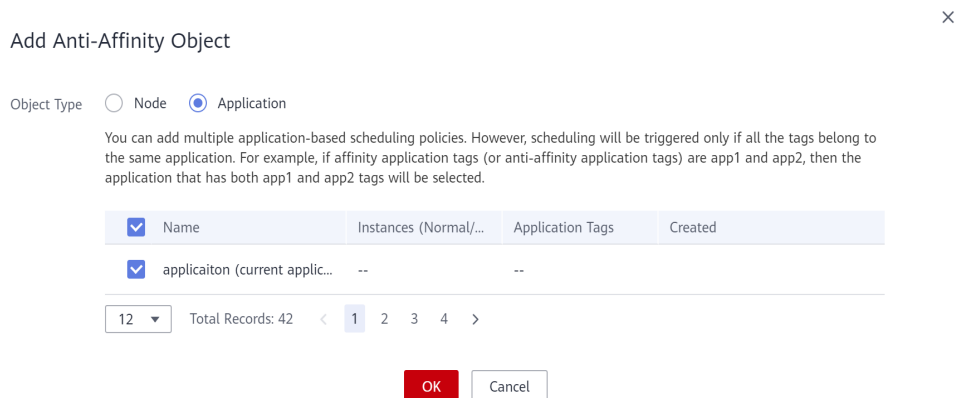
**Step 1** Click **Add Anti-Affinity Object**.

**Figure 4-47** Adding an anti-affinity object



**Step 2** In the dialog box displayed, select one or more anti-affinity nodes or applications and click **OK**.

**Figure 4-48** Selecting an anti-affinity object



----End

### 4.3.4 ConfigMaps

ConfigMaps store configuration details required for workloads. ConfigMaps decouple configuration files from container images to enhance workload portability.

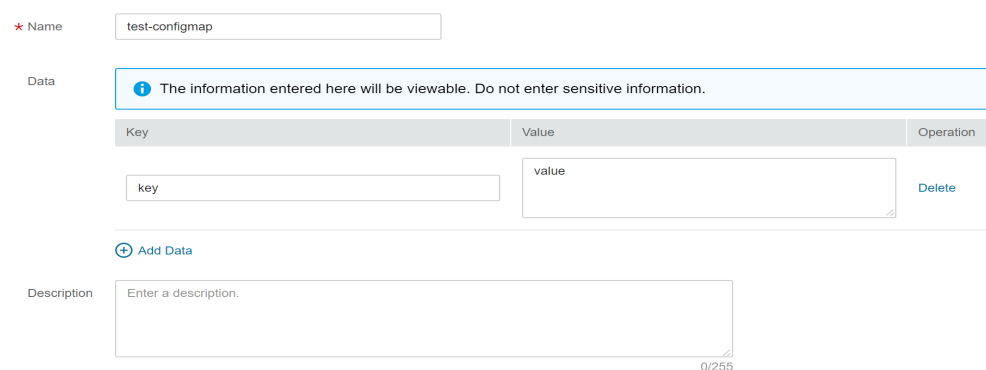
ConfigMaps allow you to:

- Manage configurations for different environments and services.
- Deploy workloads in different environments. Multiple versions are supported for configuration files so that you can update and roll back workloads easily.
- Quickly import configurations in the form of files to containers.

### Creating a ConfigMap

- Step 1** Log in to the IEF console, and click **Switch Instance** on the **Dashboard** page to select a platinum service instance.
- Step 2** In the navigation pane, choose **Edge Applications > Application Configuration**. Then, click **Create ConfigMap** in the upper right corner.
- Step 3** Specify the ConfigMap details.

**Figure 4-49** Creating a ConfigMap



- **Name:** Enter a ConfigMap name.
- **Data:** The configuration data is key-value pairs. Enter a property name and value.

**Step 4** Click **Create**. After the ConfigMap is created, the ConfigMap list page is displayed.

----End

## Using a ConfigMap

You can use ConfigMaps to configure data storage in the advanced settings when creating a containerized application.

**Figure 4-50** Using a ConfigMap

Local Volume Name	Type	Mount Directory	Permission ?	Operation
key	configMap	new-config	Read o...	420

⊕ Add Volume

After the ConfigMap is mounted to the container, files are created in the mount directory based on the ConfigMap content. Each property name and value pair is generated as a file. The property name is the file name, and the value is the content of the file. For instance, a ConfigMap could have the property name set to **key** and the property value set to **value**. After the ConfigMap is mounted to the **/tmp0** directory, a file named **key** is generated in the **/tmp0** directory and the file content is **value**.

### 4.3.5 Secrets

Secrets store sensitive, user-defined information such as authentication details, certificates, and keys, and can then be loaded to containerized applications. For example, you can mount a volume of the secret type for data store to make it a file in the container, or load a secret to make it an environment variable in the container.

#### NOTICE

Secrets may involve sensitive user information. If the secrets you attempt to configure contain sensitive information, you need to encrypt them before entering them and also need to decrypt them through applications.

## Creating a Secret

**Step 1** Log in to the IEF console, and click **Switch Instance** on the **Dashboard** page to select a platinum service instance.

**Step 2** In the navigation pane, choose **Edge Applications > Application Configuration**. Then, click **Create Secret** in the upper right corner.

**Step 3** Specify the secret details.

**Figure 4-51** Creating a secret

\* Name:

Type:

Data: Any data here should be first encoded using base64 format, and you need to ensure that the application is able to decode it.

* Key	Value	Operation
<input type="text" value="key"/>	<input type="text" value="ZXhhbXBsZSB2YWx1ZQ=="/>	<a href="#">Delete</a>

+ Add Data

Description:

- **Name:** name of the secret.
- **Type: Opaque** secrets are supported. Data is in the key-value pair format and Base64-encoded. For details, see [Base64 Encoding](#).
- **Data:** Key-value pairs. Enter a property name and value. The value must be Base64-encoded.

**Step 4** Click **Create**. After the secret is created, the secret list page is displayed.

----End

## Base64 Encoding

To perform Base64 encoding on a character string, run the **echo -n {Content to be encoded} | base64** command. For example:

```
root@ubuntu:~# echo -n "example value" | base64
ZXhhbXBsZSB2YWx1ZQ==
```

## Using a Secret

You can use secrets to configure data storage in the advanced settings when creating a containerized application.

**Figure 4-52** Using a secret

Data Storage: A local volume can be mounted to a container for persistent storage of data files.

Note: The /etc/localtime file is automatically mounted by default. Do not mount it again. Otherwise, the application creation will fail.

Local Volume Name	Type	Mount Directory	Permission	Operation
<input type="text" value="key"/>	<input type="text" value="secret"/>	<input type="text" value="sc001"/>	<input type="text" value="/tmp0"/>	<input type="text" value="Read o..."/> <input type="text" value="420"/> <a href="#">Delete</a>

+ Add Volume



After the secret is mounted to the container, files are created in the mount directory based on the secret content. Each property name and value pair is generated as a file. The property name is the file name, and the value is the content of the file. For instance, a secret could have the property name set to **key** and the property value set to **ZXhhbXBsZSB2YWx1ZQ==**. After the secret is mounted to the **/tmp0** directory, a file named **key** is generated in the **/tmp0** directory and the file content is **example value**.

## 4.3.6 Encryption Data

Encryption data is used to store and encrypt sensitive information. Edge applications can access plaintext data through MQTT server.

### Creating Encryption Data

- Step 1** Log in to the IEF console, and click **Switch Instance** on the **Dashboard** page to select a platinum service instance.
- Step 2** In the navigation pane, choose **Edge Applications > Application Configuration**. Then, click **Create Encryption Data** in the upper right corner. On the page that is displayed, specify the parameters.

**Figure 4-53** Creating encryption data

Service Instance

\* Name

\* Encryption Items ⓘ At least one item must be added. The information entered here will be viewable. Do not enter sensitive information.

Key	Value	Operation
<input type="text" value="key"/>	<input type="text" value="....."/>	<input type="button" value="Delete"/>
	<input type="text" value="....."/>	

+ Add Encryption Item

Description

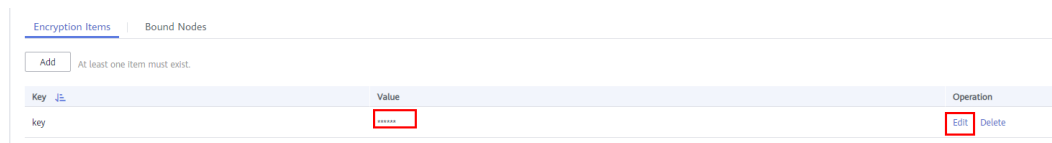
0/255

The encryption data is stored in key-value pairs. The value needs to be entered twice. Multiple encryption items can be added to each encryption data record.

- Step 3** Click **Create**.

You can view encryption data after creating it. However, for data privacy protection, data cannot be viewed in plaintext. You can also edit or delete encryption data.

**Figure 4-54** Encryption data



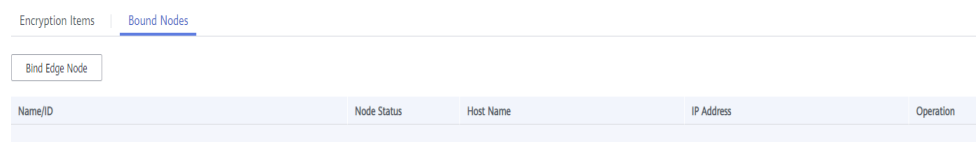
----End

## Binding Encryption Data to Edge Nodes

Encryption data can be bound to edge nodes that are not in the running state. When an edge node is restored to **Running**, the bound data will be automatically synchronized to the node. You can bind encryption data to an edge node in either of the following ways:

- On the encryption data details page, click **Bind Node** and then **Bind Edge Node**.

**Figure 4-55** Binding an edge node



- On the edge node details page, click the **Configuration** tab. In the **Encryption Data** area, click **Bind**.

**Figure 4-56** Binding encryption data to an edge node



## Using Encryption Data

After encryption data is bound to an edge node, you can obtain the data by using the MQTT client on the edge node.

A certificate must be used for security authentication when encryption data is requested. For details about the authentication method, see [Performing Security Authentication Using Certificate](#).

- Step 1** Subscribe to the topic described in [Encryption Data Acquisition](#).

**Topic:** \$hw/{project\_id}/encryptdatas/{encryptdata\_name}/properties/{properties\_name}/plaintext

- Step 2** Publish the topic described in [Encryption Data Request](#).

**Topic:** \$hw/{project\_id}/encryptdatas/{encryptdata\_name}/properties/{properties\_name}/decrypt

After the request is published, the decrypted data is sent the topic subscribed to in [Step 1](#).

----End

## 4.3.7 Health Check Configuration

Health check regularly checks the status of containers or workloads. There is a liveness probe and a readiness probe:

- **Liveness Probe:** The system executes the probe to check if a container is still alive, and restarts the instance if the probe fails. Currently, the system probes a container by HTTP request or command and determines whether the container is alive based on the response from the container.
- **Readiness Probe:** The system invokes the probe to determine whether the instance is ready. If the instance is not ready, the system does not forward requests to the instance.

IEF can perform a health check by HTTP request or through a CLI.

### HTTP Request-based Check

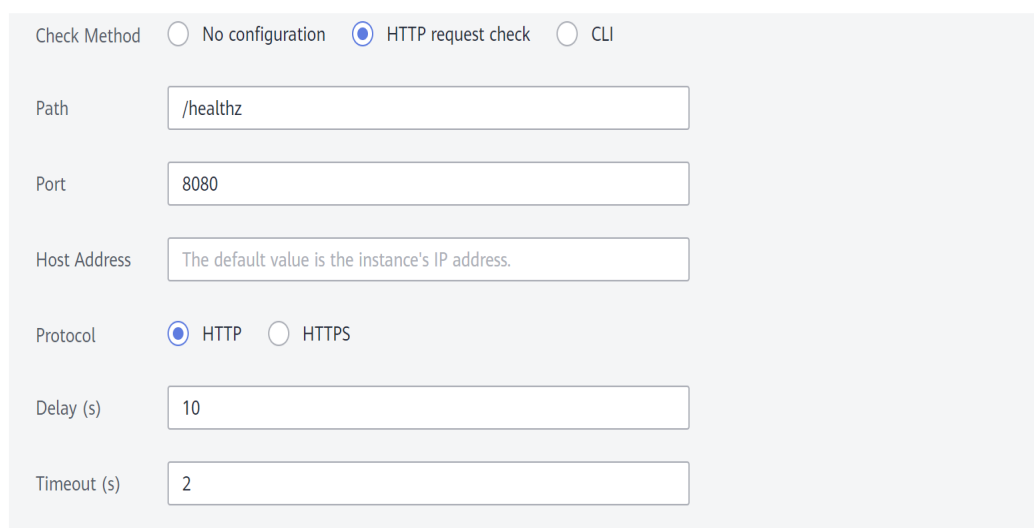
IEF periodically initiates an HTTP GET request to a container. If HTTP code 2xx or 3xx is received, the container is healthy.

For example, if health check parameters are set as shown in the following figure, IEF will send an HTTP GET request to **http://{Instance's IP address}/healthz:8080** 10 seconds after the container starts. If no response is received within 2 seconds, the health check fails. If status code 2xx or 3xx is received, the container is healthy.

#### NOTE

You do not need to specify the host address. By default, the instance's IP address is used (that is, requests are sent to the container) unless you have special requirements.

**Figure 4-57** HTTP request-based check



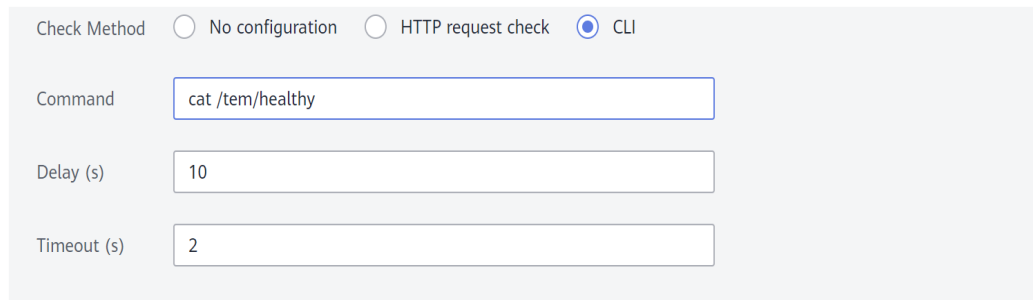
Check Method	<input type="radio"/> No configuration	<input checked="" type="radio"/> HTTP request check	<input type="radio"/> CLI
Path	<input type="text" value="/healthz"/>		
Port	<input type="text" value="8080"/>		
Host Address	<input type="text" value="The default value is the instance's IP address."/>		
Protocol	<input checked="" type="radio"/> HTTP <input type="radio"/> HTTPS		
Delay (s)	<input type="text" value="10"/>		
Timeout (s)	<input type="text" value="2"/>		

## CLI-based Check

The probe runs commands in the container and checks the command output. If the command output is 0, the container is healthy.

For example, if health check parameters are set as shown in the following figure, IEF will run **cat /tmp/healthy** 10 seconds after the container starts. If no response is received within 2 seconds, the health check fails. If the command output is 0, the container is healthy.

**Figure 4-58** CLI-based check



Check Method  No configuration  HTTP request check  CLI

Command

Delay (s)

Timeout (s)

## 4.4 Application Mesh

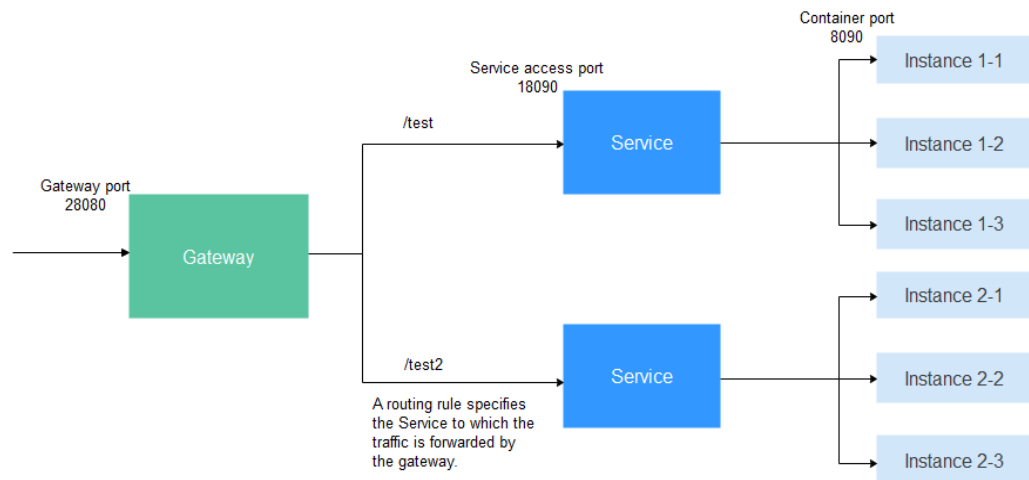
### 4.4.1 Scenario

Application Mesh is a non-intrusive microservice governance solution that supports full-lifecycle management and traffic management such as load balancing.

Application Mesh involves Services and gateways:

- **Services:** A Service defines a set of instances and a means of accessing them. The Service name can be resolved to the corresponding IP address through which applications on edge nodes can access each other.
- **Gateways:** A gateway exposes edge applications deployed using IEF to external applications and forwards traffic through virtual services.

As shown in the following figure, if you access *{IP address of the edge node where the gateway application is located}:{Gateway port}/URI*, the traffic can be forwarded to the corresponding Service, which then forwards the traffic to the corresponding backend application instance.

**Figure 4-59** Gateway traffic forwarding path

Built on the KubeEdge ecosystem, IEF extends the orchestration capabilities of cloud native containerized applications to the edge. However, in edge computing scenarios, the network topology is complex, and edge nodes in different regions usually are not connected with each other. To meet the requirement of traffic interworking between applications, IEF is compatible with community application meshes and can deliver Service and gateway metadata. Before using these functions, you need to deploy the community application EdgeMesh on edge nodes. For details, see <https://github.com/kubeedge/edgemes/>.

## 4.4.2 Services

Application Mesh is used to manage Services. When creating a Service, you can bind an application instance to it and configure an access port to implement mutual access between applications on the node.

### Creating a Service

#### NOTE

To ensure the created Service on a platinum instance of version v3 can be accessed, Edge DNS should be enabled for the containerized application that need to access the Service, and edgemes should be installed on both the node initiating the access request and the node to be accessed.

**Step 1** Log in to the IEF console, and click **Switch Instance** on the **Dashboard** page to select a platinum service instance.

**Step 2** In the navigation pane, choose **Application Mesh > Service List**. On the page displayed, click **Create Service** in the upper right corner.

**Step 3** Set the parameters.


- **Service Name:** Enter a Service name.
- **Application:** Select the application that the Service is bound to.
- **Port Settings**
  - **Access Port:** indicates the port used to access the containerized application.

- **Container Port:** indicates the port on which the containerized application listens.
- **Protocol:** Specify the protocol that will be used to access the workload. Select **HTTP** or **TCP**.

**Figure 4-60** Creating a Service

Service Name

Enter a maximum of 64 characters starting with a letter and ending with a letter or digit. Only lowercase letters, digits, and hyphens (-) are allowed.

Application  cfd-1217-001 Host network [Change](#)

Access Port	Container Port	Protocol	Operation
<input type="text" value="18090"/>	<input type="text" value="8090"/>	HTTP	Delete

[+ Add Port](#)

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

After the creation is complete, you can view the internal access domain name of the Service in the Service list.

**Figure 4-61** Viewing the internal access domain name and access port

<input type="checkbox"/> Service Name	Internal Access Domain Name	Associated Application	Access Port -> Container Port/Protocol	Updated <small>EF</small>	Operation
<input type="checkbox"/> test	<span style="border: 1px solid red; padding: 2px;">test-478acc02650141e465cabe898a39f8b.svc.cluster.local</span>	nginx-1	<span style="border: 1px solid red; padding: 2px;">8081</span> -80/HTTP	Nov 30, 2022 11:20:54 GMT+08:00	<a href="#">Update</a>   <a href="#">Traffic Policy</a>   <a href="#">Delete</a>


----End

## Updating a Service

You can update the port settings of a Service, including the access port, container port, and protocol.

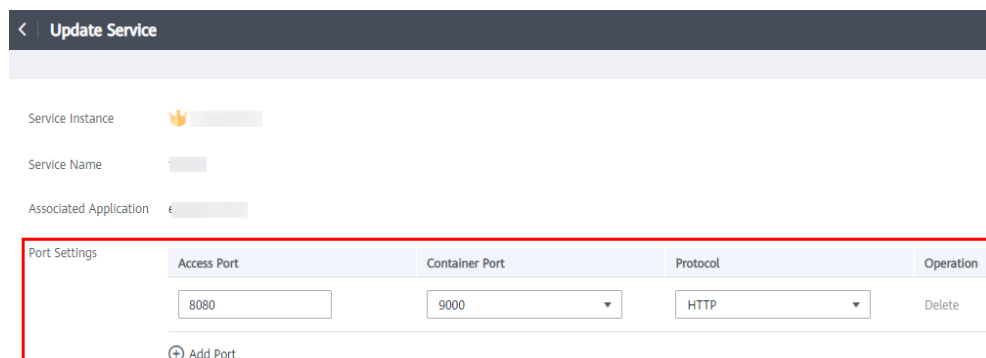
- Step 1** Log in to the IEF console, and click **Switch Instance** on the **Dashboard** page to select a platinum service instance.
- Step 2** In the navigation pane, choose **Application Mesh > Service List**. Click **Update** at the row where the Service resides.

**Figure 4-62** Updating a Service

<input type="checkbox"/> Service Name	Internal Access Domain Name	Associated Application	Access Port -> Container Port/Protocol	Operation
<input type="checkbox"/> service	 service.05e1aef9040010e22fcc...	nginx-2	18090->8090/HTTP	<span style="border: 1px solid red; padding: 2px;">Update</span>   <a href="#">Traffic Policy</a>   <a href="#">Delete</a>

- Step 3** Only **port settings** can be updated.

**Figure 4-63** Updating the port settings of a Service



**Step 4** Click **Update**.

----End

## Configuring a Traffic Policy

A traffic policy defines how service traffic is forwarded to multiple application instances.

**Step 1** Log in to the IEF console, and click **Switch Instance** on the **Dashboard** page to select a platinum service instance.

**Step 2** In the navigation pane, choose **Application Mesh > Service List**. Click **Traffic Policy** at the row where the Service resides.

**Figure 4-64** Service list

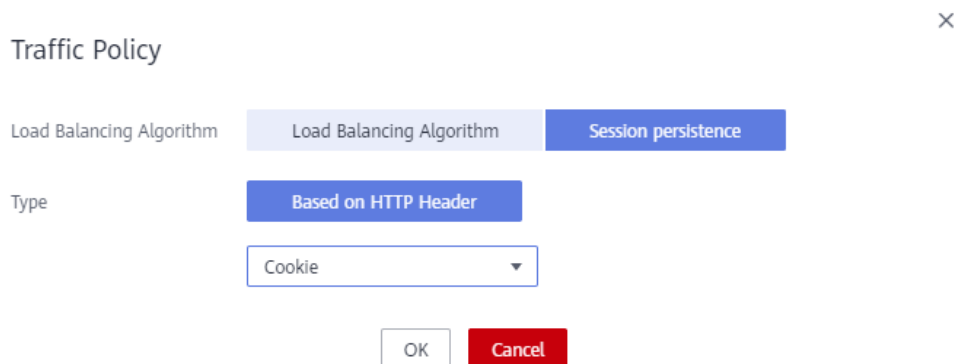
Service N...	Internal Access Domain Name	Associated Application	Access Port -> Container Port/Protocol	Updated	Operation
test001	test001.ed202955e11144468ce21a1b475f5c59.svc.dust...	easyapp-multi	8080->9000/HTTP	Jun 08, 2022 15:27:02 GMT+08:00	Update <b>Traffic Policy</b> Delete

**Step 3** On the page displayed, select a traffic policy type.

Currently, the following traffic policy types are supported:

- **Load Balancing Algorithm:** Two forwarding modes, that is, **ROUND\_ROBIN** and **RANDOM**, are supported.
- **Session persistence:** Three HTTP header-based forwarding modes, that is, **Cookie**, **User-Agent**, and **Custom**, are supported.

**Figure 4-65** Selecting a traffic policy



 **NOTE**

You can select **Load Balancing Algorithm** or **Session Persistence** for Services using HTTP. However, Services using TCP support only the load balancing algorithm.

**Step 4** Click **OK**.

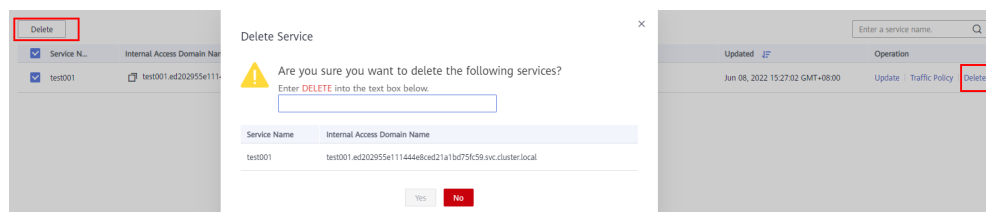
----End

## Deleting a Service

**Step 1** Log in to the IEF console, and click **Switch Instance** on the **Dashboard** page to select a platinum service instance.

**Step 2** In the navigation pane, choose **Application Mesh > Service List**. Click **Delete** at the row where the Service to be deleted resides, or select the Service and click **Delete** above the Service list.

**Figure 4-66** Deleting a Service



**Step 3** In the displayed dialog box, enter **DELETE** in the text box and click **Yes**.

----End

## 4.4.3 Gateways

Application Mesh provides gateway management. Then, the Service forwards the traffic to the corresponding backend application instance.

### Creating a Gateway

 **NOTE**

To ensure that the created gateway on a platinum instance of version v3 can be accessed, edgimesh-gateway should be installed on the node where the gateway application is located.

A gateway enables unified entry, traffic management, security, and service isolation. You can create a gateway as follows:

**Step 1** Log in to the IEF console, and click **Switch Instance** on the **Dashboard** page to select a platinum service instance.

**Step 2** In the navigation pane, choose **Application Mesh > Service Gateway**. On the page displayed, click **Create Gateway** in the upper right corner.

**Step 3** Specify the gateway information.

- **Gateway Name:** Enter a gateway name.
- **Application:** Select the gateway application.



 **NOTE**

Select the gateway application for platinum instances of version v2. For platinum instances of version v3, skip this step if edgemesh-gateway has been installed on the edge node where the gateway application is located.

- **Port Configuration**

**Port:** Enter the port number used by the gateway to access backend applications.

**Protocol:** Select the protocol used by the gateway to access backend applications. HTTP and HTTPS are supported. If you select HTTPS, configure **Protocol Version**, **Certificate Secret**, and **Cipher Suite**. If you select HTTP, configure domain names.

**Protocol Version:** Select a protocol version.

**Certificate Secret:** Select an SSL certificate. Public key files are public, and private key files are obtained by applying for certificates. Both are provided by yourself.

---

**NOTICE**

You can add the SSL certificate secret required for configuring the HTTPS protocol only here.

---

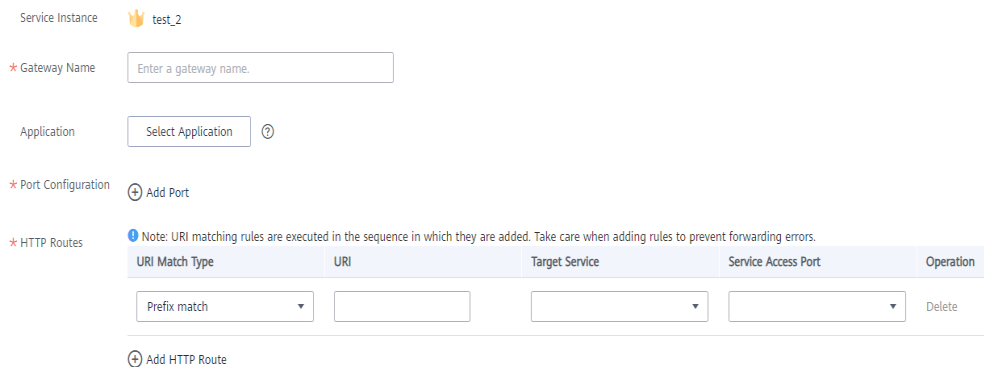
**Cipher Suite:** Select cipher suites contained in the SSL certificate and cipher suites supported by the client.


**Domain Names:** Enter the domain names that can be accessed. Regular expression matching is supported. For example, \* indicates all domain names.

- **HTTP Routes**

- **URI Match Type:** **Prefix match**, **Exact match**, and **Regular expression** are supported.
- **URI:** Enter a URI.
- **Target Service:** Select the Service to which the rule matches, that is, the Service to which the traffic from the URI is forwarded.
- **Service Access Port:** Select the port used by the Service to access the application.


**Figure 4-67** Gateway configuration




Service Instance  test\_2


\* Gateway Name

Application  ⓘ

\* Port Configuration  Add Port

\* HTTP Routes  Note: URI matching rules are executed in the sequence in which they are added. Take care when adding rules to prevent forwarding errors.

URI Match Type	URI	Target Service	Service Access Port	Operation
Prefix match	<input type="text"/>	<input type="text"/>	<input type="text"/>	Delete

 Add HTTP Route

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

----End

## Updating a Gateway

You can update the port configuration and HTTP routing rules of the service gateway.

**Step 1** Log in to the IEF console, and click **Switch Instance** on the **Dashboard** page to select a platinum service instance.

**Step 2** In the navigation pane, choose **Application Mesh > Service Gateway**. Click **Update** at the row where the gateway resides.

**Figure 4-68** Updating a gateway

Gateway Name	Associated Application	Ports	Updated	Operation
wx-gateway001	wx-reseller-s65-001	1	2022/11/28 19:41:02 GMT+08:00	Update Configure Route Delete

**Step 3** **Gateway information** other than **Gateway Name** and **Application** can be updated in this step.

**Figure 4-69** Updating gateway information

Service Instance contrc

\* Gateway Name wx-gateway005

Application

Port Configuration

\* Port  \* Protocol

\* Domain Names

\* HTTP Routes

Note: URI matching rules are executed in the sequence in which they are added. Take care when adding rules to prevent forwarding errors.

URI Match Type	URI	Target Service	Service Access Port	Operation
Prefix match	/	wx-svc003	8080->8080/HTTP	Delete

**Step 4** Click **Update**.

----End

## Configuring a Route

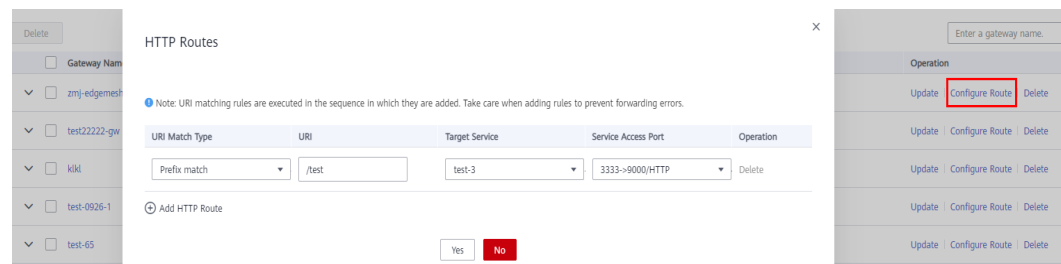
You can add multiple routes and configure multiple forwarding policies for a created gateway.

**Step 1** Log in to the IEF console, and click **Switch Instance** on the **Dashboard** page to select a platinum service instance.

**Step 2** In the navigation pane, choose **Application Mesh > Service Gateway**. Click **Configure Route** at the row where the gateway resides.

**Step 3** **Configure HTTP routes** in the displayed dialog box, and click **Yes**. The Service of the virtual service is updated.

**Figure 4-70** Configuring a route

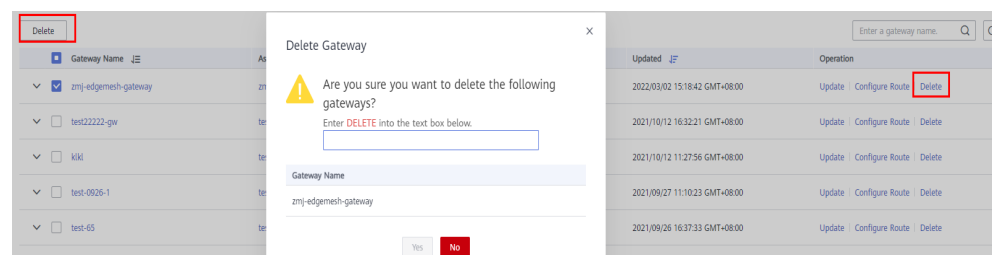


----End

## Deleting a Gateway

- Step 1** Log in to the IEF console, and click **Switch Instance** on the **Dashboard** page to select a platinum service instance.
- Step 2** In the navigation pane, choose **Application Mesh > Service Gateway**. Click **Delete** at the row where the gateway to be deleted resides, or select the gateway and click **Delete** above the gateway list.

**Figure 4-71** Deleting a gateway



- Step 3** In the displayed dialog box, enter **DELETE** in the text box and click **Yes**.

----End

## 4.5 Edge-Cloud Messages

### 4.5.1 Edge-Cloud Message Overview

IEF provides the function of routing messages exchanged between the edge and cloud. Based on configured routes, IEF forwards messages to the corresponding endpoint. In this way, messages can be forwarded based on specified paths, enhancing flexibility in data routing control and improves data security.

- Message endpoint: a party that sends or receives a message. It can be an end device or a cloud service.
- Message route: a message forwarding path.

### Message Endpoints

IEF provides the following default message endpoints:

- **SystemEventBus:** MQTT broker on an edge node, which can communicate with other endpoints on behalf of the edge node. It can function as a source endpoint to send data to the cloud, or as a destination endpoint to receive messages from the cloud. MQTT topics on the edge node are used as endpoint resources of the MQTT broker.
- **SystemREST:** a REST gateway interface in the cloud. It can function as a source endpoint to send REST requests to the edge. REST request paths are used as endpoint resources of SystemREST.

You can also create the following message endpoints:

- **Service Bus:** an endpoint deployed on an edge node to process transaction requests. It can function as a destination endpoint to process file upload requests. REST request paths are used as endpoint resources of Service Bus.
- **DIS:** data injection service. It can function as a destination endpoint to receive data forwarded by IEF. DIS streams created in DIS are used as endpoint resources.
- **API Gateway:** API gateway service. It can function as a destination endpoint to receive data forwarded by IEF. API URLs created in API Gateway are used as endpoint resources.

## Routes

Currently, IEF supports the following message forwarding paths:

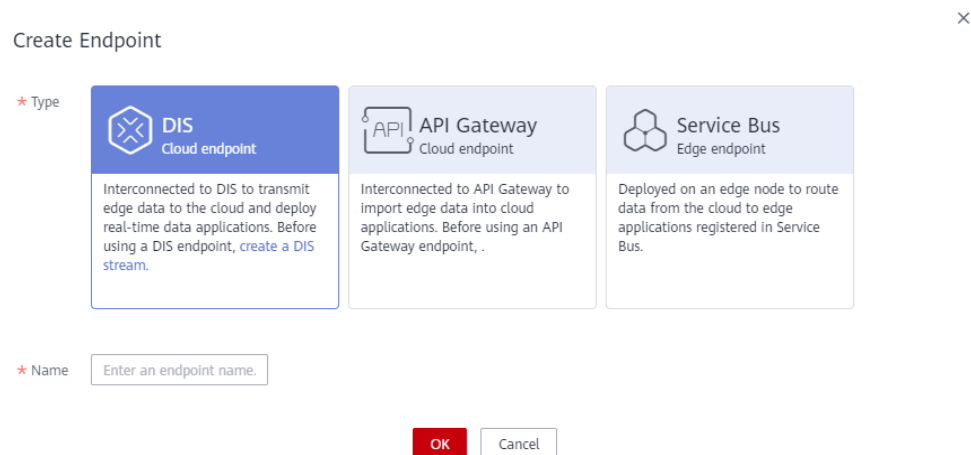
- SystemREST -> Service Bus: The REST gateway interface in the cloud is called to obtain file services on edge nodes.
- SystemREST -> SystemEventBus: The REST gateway interface in the cloud is called to send messages to SystemEventBus (MQTT broker) on edge nodes.
- SystemEventBus -> DIS/API Gateway: You can publish end device data to a custom topic in the MQTT broker of an edge node. IEF forwards the device data to a DIS stream or an API Gateway address. Then, you can extract the data for processing and analysis. You must customize an MQTT topic when creating the route. For details about custom topics, see [Custom Topics](#).

To use the message routing function, [create endpoints](#) and then [create routes](#).

## Creating an Endpoint

- Step 1** Log in to the IEF console, and click **Switch Instance** on the **Dashboard** page to select a platinum service instance.
- Step 2** In the navigation pane, choose **Edge-Cloud Messages > Endpoints**.
- Step 3** Click **Create Endpoint** in the upper right corner, and set the endpoint parameters.

**Figure 4-72** Creating an endpoint



- **Type:** Select an endpoint type. **DIS**, **API Gateway**, and **Service Bus** are supported.
- **Name:** Enter an endpoint name.
- **Service Port:** Available only for endpoints of the **Service Bus** type. Ranges from 1 to 65535.

**Step 4** Click **OK**. The endpoint is successfully created and the endpoint list page is displayed.

----End

## Creating a Route

**Step 1** Log in to the IEF console, and click **Switch Instance** on the **Dashboard** page to select a platinum service instance.

**Step 2** In the navigation pane, choose **Edge-Cloud Messages > Routes**.

**Step 3** Click **Create Route** in the upper right corner.

**Step 4** Set the route parameters.

- **Name:** Enter a route name.

---

**CAUTION**

Message routes and **system subscriptions** are of the same resource type. Their names cannot conflict with each other.

---

- **Source Endpoint:** Select a source endpoint, for example, **SystemREST** or **SystemEventBus**.
- **Source Endpoint Resource:** Select a source endpoint resource.
  - When **SystemREST** is selected:  
Enter the REST path, for example, **/abc/ab**.
  - When **SystemEventBus** is selected:

**Custom topic:** Select a node and enter a topic.

**Device data upload channel:** Select a node and select the MQTT devices bound to the node.

 **NOTE**

You can only select MQTT devices.

- **Destination Endpoint:** Select a destination endpoint, for example, **SystemEventBus**.
- **Destination Endpoint Resource:** Select a destination endpoint resource.

**Step 5** Click **Create**. The route is successfully created and the route list page is displayed.

IEF forwards messages sent to the specified resource of the source endpoint to the specified resource of the destination endpoint based on the route.

----End

## Disabling or Enabling a Route

- To disable a route, click **Disable** in the row where the route resides.  
After the route is disabled, IEF will not forward messages destined for the specified resource of the source endpoint.
- To enable a route, click **Enable** in the row where the route resides.  
After the route is enabled, IEF will forward messages destined for the specified resource of the source endpoint.

## 4.5.2 Cloud Delivering Messages to Edge Nodes

### Scenario

IEF can deliver messages from SystemREST in the cloud to SystemEventBus (MQTT broker) on edge nodes.

To be specific, IEF calls the open REST gateway interface to send messages to SystemEventBus on an edge node based on the node ID and custom topic. Your end devices can receive messages from the cloud by subscribing to custom topics.

The procedure is as follows:

1. [Creating a Route](#)
2. [Sending a Message](#)
3. [Receiving a Message](#)

### Creating a Route

SystemREST and SystemEventBus are default endpoints and do not need to be created. SystemREST indicates the REST gateway interface of IEF in the region. SystemEventBus indicates the MQTT broker of an edge node.

**Step 1** Log in to the IEF console, and click **Switch Instance** on the **Dashboard** page to select a platinum service instance.

**Step 2** In the navigation pane, choose **Edge-Cloud Messages > Routes**.

**Step 3** Click **Create Route** in the upper right corner.

**Step 4** Set parameters.

**Figure 4-73** Creating a route

The screenshot shows a form for creating a route with the following fields and values:

- Name:** SystemREST2SystemEventBus
- Source Endpoint:** SystemREST
- Source Endpoint Resource:** /
- Destination Endpoint:** SystemEventBus
- Destination Endpoint Resource:** /
- Description:** Enter a description. (0/255)

- **Name:** Enter a route name, for example, **SystemREST2SystemEventBus**.

**CAUTION**

Message routes and **system subscriptions** are of the same resource type. Their names cannot conflict with each other.

- **Source Endpoint:** Select **SystemREST**.
- **Source Endpoint Resource:** Enter a URL path starting with a slash (/). You can use {} for fuzzy match. For example, **/aaa/{any-str}/bbb** can match **/aaa/cc/bbb** or **/aaa/ddd/bbb**. The specified source resource is used as a matching field for calling the REST interface.
- **Destination Endpoint:** Select **SystemEventBus**.
- **Destination Endpoint Resource:** Enter the topic name prefix used when the message is forwarded to the MQTT broker.

**NOTE**

If both **Source Endpoint Resource** and **Destination Endpoint Resource** are set to /, IEF will forward all messages sent to the REST interface to the MQTT broker of the corresponding edge nodes. The topic name contained in the messages is the same as the request URL.

**Step 5** Click **Create**.

The created route is displayed in the message route list.

**Figure 4-74** Routes

<input type="checkbox"/>	Route Name	Source Endpoint	Destination Endpoint	Status	Forwarded Messages
<input type="checkbox"/>	SystemREST2SystemEv...	SystemREST Cloud	SystemEventBus Edge	Enabled	Total: 0 Successful: 0 Failed: 0
	--	/	/		

----End

## Sending a Message

To deliver a message from SystemREST in the cloud to SystemEventBus on an edge node, obtain the SystemREST endpoint property, construct a request, and send the request to SystemEventBus through a route.

**Step 1** Obtain the SystemREST endpoint property.

1. Log in to the IEF console, and click **Switch Instance** on the **Dashboard** page to select a platinum service instance.
2. In the navigation pane, choose **Edge-Cloud Messages > Endpoints**. In the row where SystemREST is located, the value of **public** in the **Property** column is the endpoint of SystemREST, as shown in the following figure.

**Figure 4-75** SystemREST endpoint property

<input type="checkbox"/>	Name	Type	Position	Property
<input type="checkbox"/>	SystemEventBus	eventbus (Default)	Edge	
<input type="checkbox"/>	SystemREST	rest (Default)	Cloud	public = restep.ief2.cn-north-4.myhuaweicloud.com

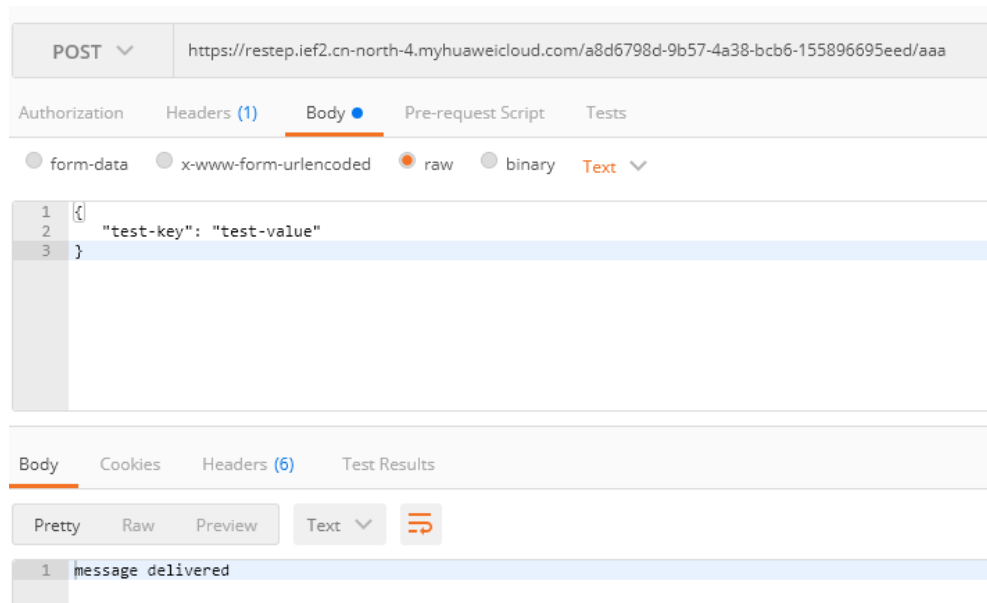
**Step 2** Construct a request and send it.

An HTTPS request needs to be constructed and sent to SystemREST. The details are as follows:

- Method: POST
- URL: **https://{rest\_endpoint}/{node\_id}/{topic}**. In the URL, *{rest\_endpoint}* is the endpoint obtained in **Step 1**, *{node\_id}* is the edge node ID, and *{topic}* is the message topic, that is, the source endpoint resource defined in **Creating a Route**.
- Body: content of the message to be sent, which is user-defined.
- Header: **X-Auth-Token**, which is a valid token obtained from IAM. For details on how to obtain a token, see **Obtaining a User Token**.

For example, use Postman to send the following message:



**Figure 4-76** Sending a message

----End

## Receiving a Message

**Step 1** Log in to the edge node.

**Step 2** Use the MQTT client to receive messages.

Currently, edge nodes support two types of MQTT brokers.

- Built-in MQTT broker (using port 8883): To use this type of MQTT broker, edge nodes must pass certificate authentication and subscribe to corresponding topics. For details about certificate authentication, see [Performing Security Authentication Using Certificate](#).
- External MQTT broker (using port 1883): To use this type of MQTT broker, install a third-party MQTT broker on edge nodes and enable edge nodes to subscribe to corresponding topics.

This section describes how to use an external MQTT broker, for example, Mosquitto, to receive messages. The procedure is as follows:

- In the Ubuntu OS, run the following commands to install Mosquitto:

```
apt-get install mosquitto
systemctl start mosquitto
systemctl enable mosquitto
```
- In the CentOS OS, run the following commands to install Mosquitto:

```
yum install epel-release
yum install mosquitto
systemctl start mosquitto
systemctl enable mosquitto
```

After the installation is complete, run the topic subscription command. If a message is sent after the subscription, the edge node will receive the message. In the following command, # indicates that any topic is subscribed to. You can replace # with a specified topic, for example, `/aaa` or `/bbb`.

```
[root@ief-node ~]# mosquitto_sub -t '#' -d
Client mosq-m02iwj4j2ISMw6rw sending CONNECT
```

```
Client mosq-m02iwj4j2ISMw6rw received CONNACK (0)
Client mosq-m02iwj4j2ISMw6rw sending SUBSCRIBE (Mid: 1, Topic: #, QoS: 0, Options: 0x00)
Client mosq-m02iwj4j2ISMw6rw received SUBACK
Subscribed (mid: 1): 0
Client mosq-m02iwj4j2ISMw6rw received PUBLISH (d0, q0, rQ, mQ, '/aaa', ... (31 bytes))
{
  "test-key": "test-value"
}
```

----End

## 4.5.3 Edge Nodes Reporting Messages to the Cloud

### Scenario

IEF allows edge nodes to report messages to the cloud.

You can publish data to a custom topic in SystemEventBus (MQTT broker) of edge nodes. IEF forwards the data to a DIS stream or an API Gateway address. Then, you can extract the data for processing and analysis.

This section uses the DIS endpoint as an example. The method of using the API Gateway endpoint is similar. The procedure is as follows:

1. [Creating an Endpoint](#)
2. [Buying a DIS Stream](#)
3. [Creating a Route](#)
4. [Sending a Message](#)

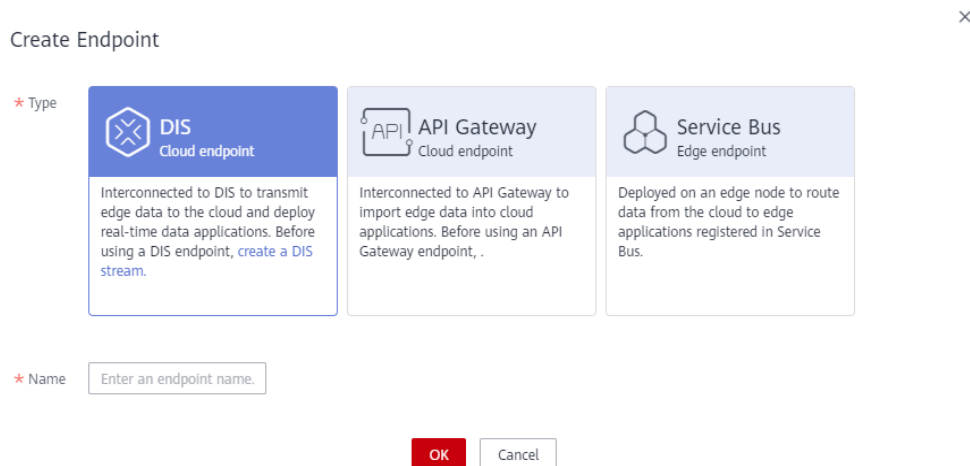
### Creating an Endpoint

**Step 1** Log in to the IEF console, and click **Switch Instance** on the **Dashboard** page to select a platinum service instance.

**Step 2** In the navigation pane, choose **Edge-Cloud Messages > Endpoints**.

**Step 3** Click **Create Endpoint** in the upper right corner. Select **DIS** for **Type**, and enter an endpoint name.

**Figure 4-77** Creating an endpoint



**Step 4** Click **OK**.

----End

## Buying a DIS Stream

Before sending messages to DIS, buy a DIS stream.

**Step 1** Log in to the DIS console.

**Step 2** Click Buy Stream in the right corner and set parameters.

**Figure 4-78** Buying a DIS stream

\* Billing Mode Pay-per-use

\* Region CN East-Shanghai1

Regions are geographic areas isolated from each other. Resources are region-specific and cannot be used across regions through internal network connections. For low network latency and quick resource access, select the nearest region.

\* Stream Name dis-ief-1

The system automatically populates an editable stream name that contains the prefix "dis" followed by four alphanumeric characters.

\* Stream Type Common Advanced ?

\* Partitions ? 1 Partition Calculator You can use a maximum of 49 partitions. [Learn how to increase quota.](#)

Selected: Common DIS stream | 1 partition | maximum stream capacity: 1 MB/s (write); 2 MB/s (read)

\* Data Retention (hours) 24

\* Source Data Type BLOB JSON CSV ?

\* Auto Scaling ?

Enterprise Project ? default C

Advanced Settings Skip Configure

**Step 3** Click **Next**, confirm the product specifications, and click **Submit**.

----End

## Creating a Route

**Step 1** Log in to the IEF console, and click **Switch Instance** on the **Dashboard** page to select a platinum service instance.

**Step 2** In the navigation pane, choose **Edge-Cloud Messages > Routes**.

**Step 3** Click **Create Route** in the upper right corner.

**Step 4** Set parameters.

**Figure 4-79** Creating a route

- **Name:** Enter a route name.

**CAUTION**

Message routes and **system subscriptions** are of the same resource type. Their names cannot conflict with each other.

- **Source Endpoint:** Select **SystemEventBus**.
- **Source Endpoint Resource:** Select **Custom topic**, select the edge node that will send messages, and enter a topic name.
- **Destination Endpoint:** Select the endpoint created in **Creating an Endpoint**.
- **Destination Endpoint Resource:** Select the DIS stream purchased in **Buying a DIS Stream**.

**NOTE**

- Record the topic, as shown in the red box in the preceding figure. After the route is created, you can view the topic in the **Source Endpoint** column of the route list.
- After customizing a topic, you need to use the complete topic (marked in red in the figure above) for messaging.

**Step 5** Click **Create**.

----End

## Sending a Message

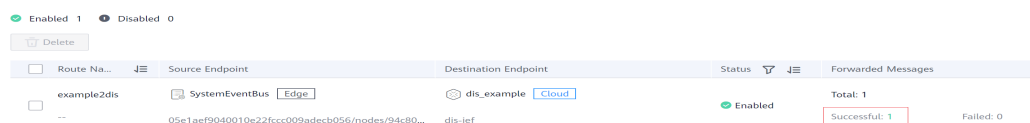
Use the MQTT client on the edge node to send messages.

In this example, the message needs to be published to the topic specified in [Creating a Route](#). As shown in the following figure, mosquitto\_pub is used to send a message.

```
[root@ief-node ~]# mosquitto_pub -t '05e1aef9040010e22fcc009adecb056/nodes/7092ad14-adee-4a09-b969-1505bbdecef5/user/aaa' -d -m '{ "edgemsg": "msgToCloud"}'
Client mosq-p5LouPQIW2gx0JPkRF sending CONNECT
Client mosq-p5LouPQIW2gx0JPkRF received CONNACK (0)
Client mosq-p5LouPQIW2gx0JPkRF sending PUBLISH (d0, q0, r0, m1, '05e1aef9040010e22fcc009adecb056/nodes/7092ad14-adee-4a09-b969-1505bbdecef5/user/aaa', ... (26 bytes))
Client mosq-p5LouPQIW2gx0JPkRF sending DISCONNECT
```

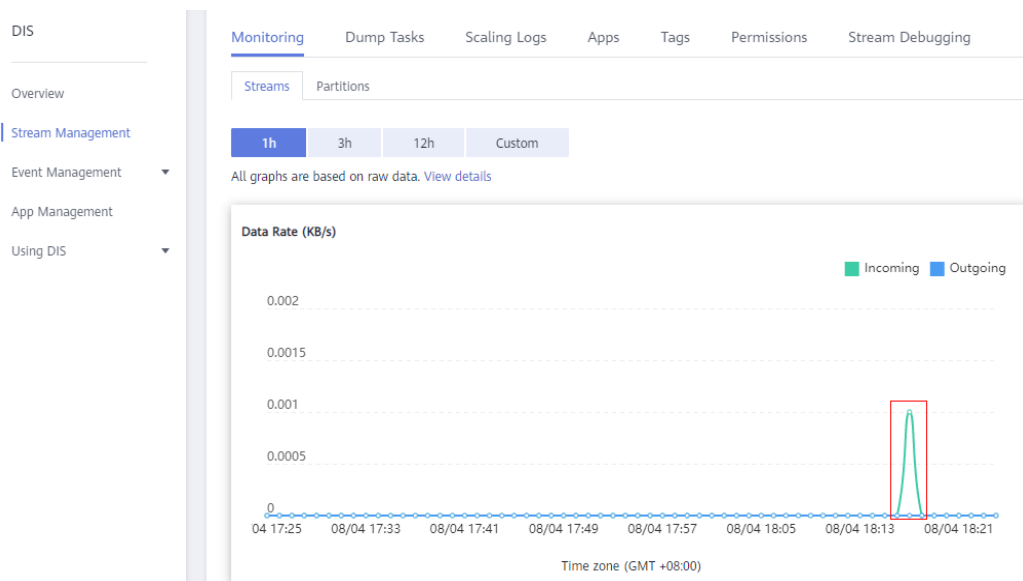
After the message is sent, you can view the **Forwarded Messages** column in the route list to check whether the message is successfully forwarded by the route.

**Figure 4-80** Number of forwarded messages



You can view incoming messages on the DIS console.

**Figure 4-81** DIS data monitoring



## Obtaining Data

After data is forwarded to DIS stream, you can extract the data for processing and analysis. For details on how to obtain data from DIS, see [Obtaining Data from DIS](#).

## 4.5.4 System Subscriptions

### Scenario

IEF provides the system subscription function for you to subscribe to IEF resource change events. When a resource is created, updated, or deleted, IEF will send a

message to the specified API Gateway endpoint so that you can obtain resource changes in a timely manner.

System subscription is the special implementation of edge-cloud messages. IEF sends event messages about specific resources to a specified topic and calls an API of API Gateway to notify you of resource changes.

## Events That Can Be Subscribed To

You can subscribe to the following events from IEF.

**Table 4-12** Events that can be subscribed to

System Event	Topic	Resource Type	Operation
Instance creation	\$hw/events/instance/+/created	Application instance	Create
Instance update	\$hw/events/instance/+/updated	Application instance	Update
Instance deletion	\$hw/events/instance/+/deleted	Application instance	Delete
Application deletion	\$hw/events/deployment/+/deleted	Containerized application	Delete
Application creation	\$hw/events/deployment/+/created	Containerized application	Create
Application update	\$hw/events/deployment/+/updated	Containerized application	Update
Node creation	\$hw/events/edgeNode/+/created	Edge node	Create
Node update	\$hw/events/edgeNode/+/updated	Edge node	Update
Node deletion	\$hw/events/edgeNode/+/deleted	Edge node	Delete
Node being brought online	\$hw/events/edgeNode/+/online	Edge node	Bring online
Node being brought offline	\$hw/events/edgeNode/+/offline	Edge node	Bring offline
Device creation	\$hw/events/device/+/created	End device	Create
Device update	\$hw/events/device/+/updated	End device	Update

System Event	Topic	Resource Type	Operation
Device deletion	\$hw/events/device/+/ deleted	End device	Delete

## System Subscription Process

The system subscription process is as follows:

1. Create an API on API Gateway for IEF to call.
2. Create an API Gateway endpoint on IEF.
3. Create a system subscription on IEF.

## Creating an API

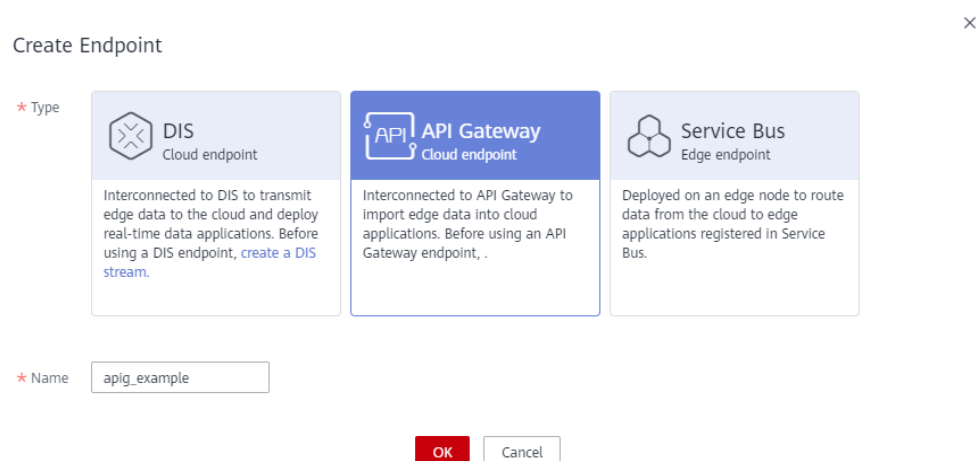
Create an API on API Gateway. For details, see [API Gateway Introduction](#).

This API is called by IEF to send system event messages.

## Creating an API Gateway Endpoint

- Step 1** Log in to the IEF console, and click **Switch Instance** on the **Dashboard** page to select a platinum service instance.
- Step 2** In the navigation pane, choose **Edge-Cloud Messages > Endpoints**.
- Step 3** Click **Create Endpoint** in the upper right corner, and set the endpoint parameters.

**Figure 4-82** Creating an endpoint



- **Type:** Select **API Gateway**.
- **Name:** Enter an endpoint name.

**Step 4** Click **OK**. The endpoint is successfully created and the endpoint list page is displayed.

----End

## Creating a System Subscription

**Step 1** Log in to the IEF console, and click **Switch Instance** on the **Dashboard** page to select a platinum service instance.

**Step 2** In the navigation pane, choose **Edge-Cloud Messages > System Subscriptions**.

**Step 3** Click **Create System Subscription** in the upper right corner, and set the system subscription parameters.

**Figure 4-83** Creating a system subscription

The screenshot shows a form for creating a system subscription. It has the following fields and values:

- Name:** subscription
- Topic:** Application (dropdown), Create (dropdown), and a plus icon (+)
- Destination Endpoint:** apig\_example (dropdown)
- Destination Endpoint Resource:** https://4e93a29712a245d2b700b70b0316ab04.apig  
The domain name in the URL must be configured in API Gateway. For details, see [Binding a Domain Name](#).
- Description:** Enter a description. (text area)

0/255

- **Name:** Enter a system subscription name.

---

**CAUTION**

System subscriptions and [message routes](#) are of the same resource type. Their names cannot conflict with each other.

- 
- **Topic:** Select the resource and operation to be subscribed, for example, creating a containerized application. For details about the events that can be subscribed to, see [Table 4-12](#).
  - **Destination Endpoint:** Select the endpoint created in [Creating an API Gateway Endpoint](#).
  - **Destination Endpoint Resource:** Select the API created in [Creating an API](#).

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

----End



## Message Forwarding After Subscription

After a system subscription is created, IEF will forward a message when a system event occurs. The statistics about message forwarding can be viewed on the **System Subscriptions** page of the IEF console.

**Figure 4-84** Number of forwarded messages

<input type="checkbox"/>	Name	Topic	Destination Endpoint	Status	Forwarded Messages
<input type="checkbox"/>	subscription	deployment/created	apig_example Cloud https://4e93a29712a245d2b700b70b0316ab0...	Enabled	Total: 0 Successful: 0 Failed: 0

Meanwhile, IEF will call the API registered on API Gateway. The request body is as follows: (The request body uses the standard CloudEvents format. For details about CloudEvents, click [here](#).)

```
{
  "data": {
    "event_type": "instance",
    "operation": "created",
    "timestamp": 134567677,
    "topic": "$hw/events/deployment/+/created",
    "name": "xxx",
    "attributes": {"ID": "x"}
  },
  "datacontenttype": "application/json",
  "source": "sysevents",
  "id": "xxx",
  "time": "2020-11-5 xxx"
}
```

- **data:** system event data.
  - **event\_type:** resource type. The value is a character string.
  - **operation:** operation performed on the resource. The value is a character string.
  - **topic:** topic to which a message is sent. The value is a character string.
  - **timestamp:** time when an event occurs. The value is of the UInt64 type.
  - **name:** resource name. The value is a character string.
  - **attributes:** resource attributes. This parameter is not contained in the request for deleting a resource. The value is of the Object type.
- **datacontenttype:** format of the system event data content. The value is a character string.
- **source:** source of the system event. The value is a character string.
- **id:** system event ID, which is a character string.
- **time:** time when the system event occurs. The value is a character string.

The following table lists the resource types, operations, and topics supported by IEF.

**Table 4-13** Events that can be subscribed to

System Event	Topic	Resource Type	Operation
Instance creation	\$hw/events/instance/+/created	Application instance	Create
Instance update	\$hw/events/instance/+/updated	Application instance	Update
Instance deletion	\$hw/events/instance/+/deleted	Application instance	Delete
Application deletion	\$hw/events/deployment/+/deleted	Containerized application	Delete
Application creation	\$hw/events/deployment/+/created	Containerized application	Create
Application update	\$hw/events/deployment/+/updated	Containerized application	Update
Node creation	\$hw/events/edgeNode/+/created	Edge node	Create
Node update	\$hw/events/edgeNode/+/updated	Edge node	Update
Node deletion	\$hw/events/edgeNode/+/deleted	Edge node	Delete
Node being brought online	\$hw/events/edgeNode/+/online	Edge node	Bring online
Node being brought offline	\$hw/events/edgeNode/+/offline	Edge node	Bring offline
Device creation	\$hw/events/device/+/created	End device	Create
Device update	\$hw/events/device/+/updated	End device	Update
Device deletion	\$hw/events/device/+/deleted	End device	Delete

## 4.6 Batch Management

## 4.6.1 Registering Nodes in Batches

### Scenario

If you need to manage, update, and maintain a large number of edge nodes of the same type, it will be labor-consuming to do so one by one as described in [Registering an Edge Node](#).

IEF provides batch node registration to pre-install required software on edge nodes of the same type so that they can be connected to the network after they are powered on. This one-to-many design improves management efficiency and reduces O&M costs.

### Procedure of Registering Nodes in Batches

The procedure for managing edge nodes in batches is as follows:

1. Prepare edge nodes that meet certain requirements. Then, configure the environment on the edge nodes by referring to [Configuring the Edge Node Environment](#).
2. After [creating a batch node registration job](#), obtain the configuration file and registration software, and pre-install them on the edge nodes. For details, see [Batch Management of Edge Nodes](#). If the registration command is configured in the startup script, the device will be automatically managed as an edge node of IEF after being powered on and connected to the network.

### Creating a Batch Node Registration Job

- Step 1** Log in to the IEF console, and click **Switch Instance** on the **Dashboard** page to select a platinum service instance.
- Step 2** In the navigation pane, choose **Batch Management > Node Registration Jobs**, and click **Register Edge Nodes** in the upper right corner of the displayed page.
- Step 3** Configure basic information. The parameters marked with \* are mandatory.

The screenshot shows a web form for creating a batch node registration job. At the top, it displays 'Service Instance' with a crown icon and the text 'cl 2'. Below this, there are several input fields:

- \* Name:** A text input field containing the character 'i'.
- Description:** A large text area with the placeholder text 'Enter a description.' and a character count '0/255' at the bottom right.
- Tags:** A section with two input fields: 'Tag key' and 'Tag value'. Below them is the text 'You can add 20 more tags.'
- Properties:** A table with three columns: 'Key', 'Value', and 'Operation'. Below the table is a button labeled 'Add Property'.

- **Name:** name of the batch node registration job.
- **Description:** description of the job.

- **Tags:** Tags can be used to classify resources, facilitating resource management.
- **Properties:** A property is in key-value pair format. Enter a key and value.

**Step 4** Click **Register** in the lower right corner. You can choose to register the edge nodes using a certificate or a token.

- Method 1: registration using a certificate: Download the configuration file, EdgeCore Register, and EdgeCore Installer.

---

**NOTICE**

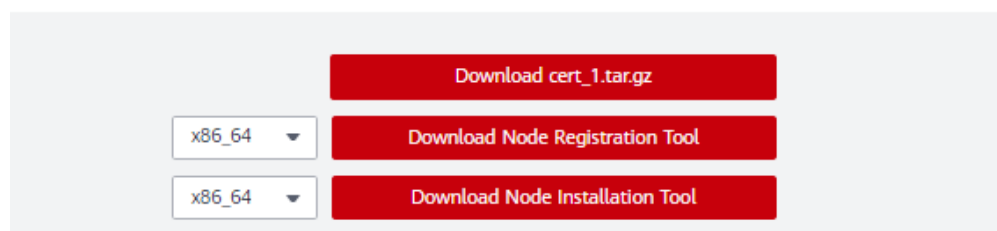
To enhance node security, you must download the configuration file and tools now. The configuration file and tools cannot be retrieved later.

---

**Figure 4-85** Downloading the configuration file, EdgeCore Register, and EdgeCore Installer

Download the product certificate and tools to complete the creation.

Download the certificate right now because you will not be able to retrieve it afterwards.



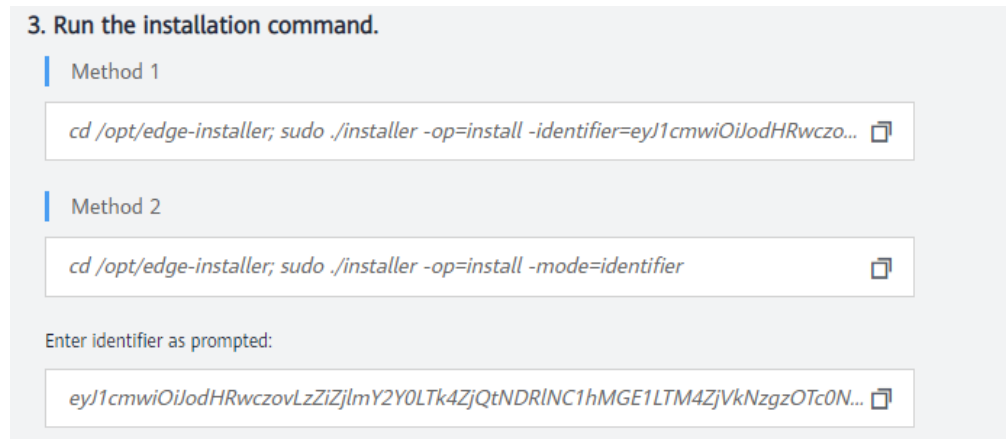
- a. Click **Download cert\_1.tar.gz** to download the configuration file.
  - b. Select the CPU architecture of your edge nodes, and click **Download Node Registration Tool** and **Download Node Installation Tool**.
- Method 2: registration using a token: Download the EdgeCore Installer and save the installation credential.

**Figure 4-86** Downloading EdgeCore Installer

Download the following configuration files and software to complete the creation.

To connect a node to an IEF, perform the following steps.



**Figure 4-87** Installation command

- a. Select the CPU architecture of your edge nodes, and click **Download EdgeCore Installer**.
- b. Commands for the two installation methods are displayed. Save either of them.

**Step 5** Upload the downloaded file to the edge nodes, and then manage the edge nodes on IEF by referring to [Batch Management of Edge Nodes](#).

----End

## Batch Management of Edge Nodes

Follow the procedure of the finish step on the **Register Edge Nodes** page to perform the batch management operation to the edge nodes.

**Step 1** Log in to an edge node as a user with sudo permissions.

**Step 2** Run the following commands to decompress the configuration file, EdgeCore Register, and EdgeCore Installer.

```
sudo tar -zxvf edge-installer_1.0.10_x86_64.tar.gz -C /opt
```

```
sudo tar -zxvf edge-register_2.0.10_x86_64.tar.gz -C /opt
```

```
sudo tar -zxvf cert_batchNodes.tar.gz -C /opt/edge-register/ (This command is not required for registration using a token.)
```

Replace `edge-installer_1.0.10_x86_64.tar.gz`, `edge-register_2.0.10_x86_64.tar.gz`, and `cert_batchNodes.tar.gz` with the names of the files downloaded in [Creating a Batch Node Registration Job](#).

**Step 3** (Optional) Customize the node information.

For batch node management, the default configuration will be applied for these edge nodes. Where:

- The node name is the combination of the host name and MAC address of the node, that is, **Host name\_MAC address**.
- The GPU and NPU options are disabled.
- The Docker function is enabled.

If you want to customize node information, configure the **nodeinfo** file. Run the following commands to open the **nodeinfo** file:

```
cd /opt/edge-register
```

```
vi nodeinfo
```

Specify the node information in the **nodeinfo** file. The following example lists all configurable parameters. You can set the parameters based on site requirements and delete the parameters that do not need to be customized.

Parameters such as the node name, descriptions, GPU enabled or not, NPU specifications, log configuration, and attributes are included. For details about the parameters, see [API Reference > Registering an Edge Node](#).

```
{
  "name": "nodename",
  "description": "",
  "enable_gpu": false,
  "enable_npu": true,
  "npu_type": "****",
  "enable_docker": true,
  "attributes": [
    {
      "key": "key1",
      "value": "value1"
    }
  ],
  "log_configs": [
    {
      "component": "app",
      "type": "local",
      "level": "debug",
      "size": 100,
      "rotate_num": 5,
      "rotate_period": "daily"
    }
  ],
  "device_infos": [
    {
      "device_ids": ["15696983-5ee6-43b4-9653-5d8512813dcc"],
      "relation": "camera",
      "comment": "devicedescription"
    }
  ],
  "mqtt_config": {
    "enable_mqtt": false,
    "mqttps": []
  },
  "tags": [
    {
      "key": "name",
      "value": "value"
    }
  ]
}
```

**Step 4** Run one of the following commands to manage the edge nodes:

- For edge nodes registered using certificates:

```
cd /opt/edge-register; ./register --mode=cert
```

- For edge nodes registered using tokens:

```
cd /opt/edge-register; ./register --mode=identifier --identifier=Credential  
for registration using a token
```

Replace *Credential for registration using a token* with the value of **identifier** field of the installation credential saved in [Creating a Batch Node Registration Job](#).

After the command is executed, check whether the node is successfully registered by referring to [Viewing Managed Nodes](#).

You can also add the registration command `cd /opt/edge-registry; ./register --mode=cert` to the startup script. In this way, the edge node can automatically run the registration command when it is powered on.

If the command output is **0**, the edge node is successfully registered on IEF.

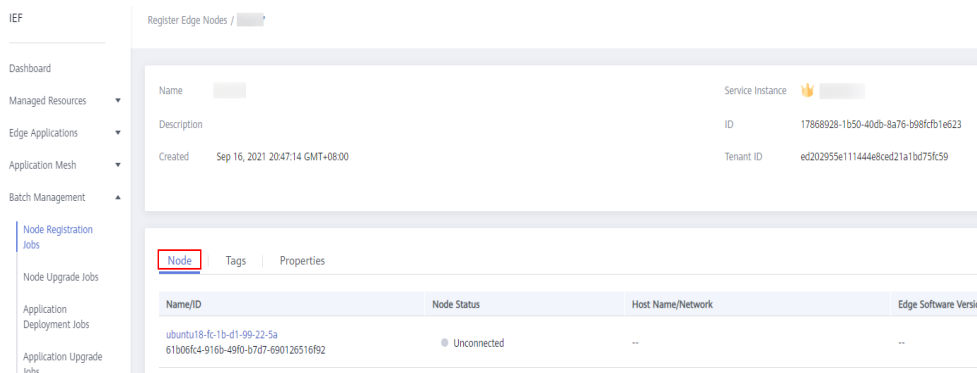
----End

## Viewing Managed Nodes

You can view the nodes managed in batches on the console.

- Step 1** Log in to the IEF console, and click **Switch Instance** on the **Dashboard** page to select a platinum service instance.
- Step 2** In the navigation pane, choose **Batch Management > Node Registration Jobs**, and click a job name on the displayed page.
- Step 3** You can view the managed nodes on the **Nodes** tab.

**Figure 4-88** Viewing nodes managed in batches



----End

## 4.6.2 Upgrading Nodes in Batches

### Scenario

In some scenarios, you may need to manage, update, and maintain EdgeCore software of a large number of edge nodes. IEF allows upgrading edge nodes in batches for this purpose.

#### NOTE

Edge nodes can be upgraded successfully only when they communicate with IEF properly.

## Precautions

- IEF does not proactively upgrade EdgeCore on your edge nodes. You are advised to upgrade the node manually in the time window with the minimum impact on your services.
- Upgrading a version within the maintenance period will not interrupt your applications running on the edge node. However, if message routing is used, services may be temporarily affected.
- Upgrading a version beyond the maintenance period may temporarily interrupt services due to container restart.
- Do not change node configurations during the node upgrade, such as restarting Docker, installing or uninstalling the GPU/NPU driver, upgrading the OS kernel, or modifying network configurations, which may result in node upgrade failures.

## Procedure

- Step 1** Log in to the IEF console, and click **Switch Instance** on the **Dashboard** page to select a platinum service instance.
- Step 2** In the navigation pane, choose **Batch Management > Node Upgrade Jobs**, and click **Create Node Upgrade Job** in the upper right corner of the displayed page.
- Step 3** Configure basic information about the batch node upgrade job.

**Figure 4-89** Upgrading nodes in batches

The screenshot shows a web form for creating a node upgrade job. It is divided into two main sections by a horizontal line. The top section contains the following fields:

- Name:** A text input field with a red asterisk icon and the placeholder text "Enter a job name."
- Description:** A large text area with the placeholder text "Enter a job description." and a character count "0/255" at the bottom right.
- Tags:** Two input fields labeled "Tag key" and "Tag value". Below them is the text "You can add 20 more tags."

The bottom section, titled "Upgrade Configuration", contains:

- Upgrade Object:** A dropdown menu with the option "Select Edge Node" selected.



- **Name:** name of the node upgrade job to be created.
- **Description:** description of the node upgrade job.
- **Tags:** tag of the node upgrade job.
- **Upgrade Object:** node to be upgraded.

Click **Select Edge Node**, and select the nodes to be upgraded.

You can also click **Search by Tag** in the upper right corner of the page, enter the tag key and value, and click **Search** to filter out the edge nodes with the specified tag. Then, select the edge nodes to be upgraded and click **OK**.

**Step 4** Click **Next**. Confirm the upgrade information and click **Create**.

----End

## Status Description

A batch node upgrade job can be in any of the following states.

- **Pending:** The job is waiting to be executed.
- **Running:** The job is being executed.
- **Successful:** All tasks in the job are executed successfully.
- **Partially successful:** Some tasks in the job are executed successfully.
- **Failed:** All tasks in the job failed.
- **Stopping:** The job is being stopped.
- **Stopped:** The job is stopped.
- **Update timed out:** The job is pended for more than 10 minutes or the job has not completed even after 10 minutes.

A job can be stopped during execution and resumed after being stopped.

If a job fails to be executed, partially succeeds, or times out, you can retry the job.

**Figure 4-90** Retry

Name/ID	Status	Results	Created	Updated	Description	Operation
we-nginx-0005 4d3377a-7564-4766-b9d8-b6f8da2a66a1	Success	Total 2 Successful 2 Failed 0 Pending 0	Aug 08, 2022 11:36:12 GMT+08:00	Aug 08, 2022 11:38:11 GMT+08:00	---	Stop   Resume   More
we-testlog-arm-0002 14a8b71b-f688-4717-8b73-25a0031b88f0	Partially su...	Total 3 Successful 2 Failed 1 Pending 0	Aug 08, 2022 11:38:45 GMT+08:00	Aug 08, 2022 11:32:36 GMT+08:00	---	Stop   Resume   More Retry Delete

## 4.6.3 Deploying Applications in Batches

### Scenario

In some scenarios, you may need to deploy the same applications on a large number of edge nodes. IEF allows deploying applications in batches for this purpose.

#### NOTE

- All edge nodes must use the same architecture, for example, x86 or Arm.
- The architecture of the container image to be used must be the same as that of the edge node on which a containerized application is to be deployed. For example, if the edge node uses x86, the container image must also use x86.

## Procedure

- Step 1** Log in to the IEF console, and click **Switch Instance** on the **Dashboard** page to select a platinum service instance.
- Step 2** In the navigation pane, choose **Batch Management > Application Deployment Jobs** and click **Create Application Deployment Job** in the upper right corner of the displayed page.
- Step 3** Specify basic information about the application deployment job.

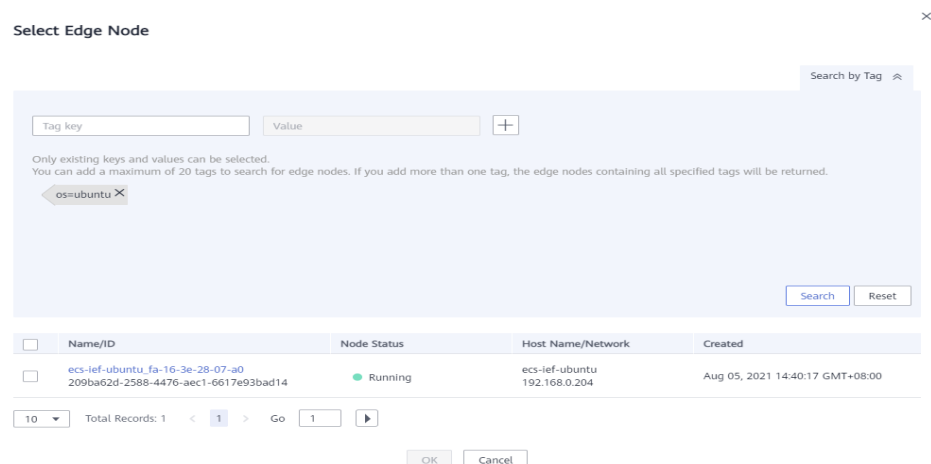
**Figure 4-91** Creating a batch application deployment job

The screenshot shows a multi-step wizard for creating a batch application deployment job. The current step is 'Specify Basic Information'. The form includes a 'Service Instance' dropdown menu, a 'Name' field (required, with a red border and placeholder 'Enter a job name.'), a 'Description' text area (placeholder 'Enter a job description.', 0/255 characters), and 'Tags' with 'Tag key' and 'Tag value' input fields. Below the 'Specify Basic Information' section is the 'Deployment Configuration' section, which includes a 'Deployment Object' dropdown menu (set to 'Manually specify') and a 'Select Edge Node' button. The 'Advanced Settings' section is expanded, showing 'Restart Policy' (set to 'Always restart') and 'Host PID' (set to 'Disable'). At the bottom right, there are 'Cancel' and 'Next' buttons.

- **Name:** name of the application deployment job to be created.
- **Description:** description of the application deployment job.
- **Tags:** tag of the application deployment job.
- **Deployment Object:** edge node where the containerized application is to be deployed.

Click **Select Edge Node**, and select target edge nodes.

You can also click **Search by Tag** in the upper right corner of the page, enter the tag key and value, and click **Search** to filter out the edge nodes with the specified tag. Then, select edge nodes and click **OK**.

**Figure 4-92** Selecting tags

- **Restart Policy**
  - **Always restart:** The system restarts the container regardless of whether it had quit normally or unexpectedly.
  - **Restart upon failure:** The system restarts the container only if it had previously quit unexpectedly.
  - **Do not restart:** The system does not restart the container regardless of whether it had quit normally or unexpectedly.
- **Host PID:** Enabling this function allows containers to use the host's PID namespace. This function can be enabled only on edge nodes whose software version is 2.8.0 or later.

**Step 4** Click **Next**. Then, configure application deployment information.

- **Name Prefix:** prefix of the application deployment name. IEF generates a complete name based on the prefix.
- **Instances:** number of instances deployed for the application.
- **Configuration Method**
  - **Custom:** Configure the application step by step. For details, see [Step 5](#).
  - **Application template:** Select a predefined application template and modify it based on your requirements. This method reduces repeated operations. The configurations in a template are the same as those set in [Step 5](#). For details on how to create a template, see [Application Templates](#).
- **Description:** description of the containerized application.
- **Tags:** tag of the containerized application. You can select **Inhibit common tags from edge nodes**.

**Step 5** Configure containers.

Click **Use Image** under the target image.

- **My Images:** shows all the images you have created in [SWR](#).
- **Shared Images:** displays images shared by other users. Shared images are managed and maintained in SWR. For details, see [Sharing a Private Image](#).

After selecting an image, specify container configurations.

- **Image Version:** Select the version of the image to be used to deploy the application.

**NOTICE**

Do not use version **latest** when deploying containers in the production environment. Otherwise, it will be difficult to determine the version of the running image and to roll back the application properly.

- **Container Specifications:** Specify the CPU, memory, Ascend AI accelerator card, and GPU quotas.
- Ascend AI accelerator card: The AI accelerator card configuration of the containerized application must be the same as that of the edge node actually deployed. Otherwise, the application will fail to be created. For details, see [Registering an Edge Node](#).

**NOTE**

For NPUs after virtualization partition, only one virtualized NPU can be mounted to a container. The virtualized NPU can be allocated to another container only after the original container quits.

AI accelerator cards support NPU types listed in the following table:

**Table 4-14** NPU types

Type	Description
Ascend 310	Ascend 310 chips
Ascend 310B	Ascend 310B chips

**Figure 4-93** Container configuration

The screenshot shows a configuration form for a container. At the top, the 'Image Name' is set to 'nginx' with a 'Change Image' link. Below it, the 'Image Version' is set to 'latest' in a dropdown menu. The 'Container Name' is 'container-75'. Under 'Container Specifications', there are sections for 'CPU' and 'Memory'. The CPU section has 'Request' and 'Limit' both set to '0.25 cores'. The Memory section has 'Request' and 'Limit' both set to '512.00 MIB'. At the bottom, there is an 'AI Accelerator Card' section with three options: 'Not installed' (selected), 'Ascend AI accelerator card', and 'NVIDIA GPU'.

You can also configure the following advanced settings for the container:

- **Command**  
A container image has metadata that stores image details. If lifecycle commands and arguments are not set, IEF runs the default commands and

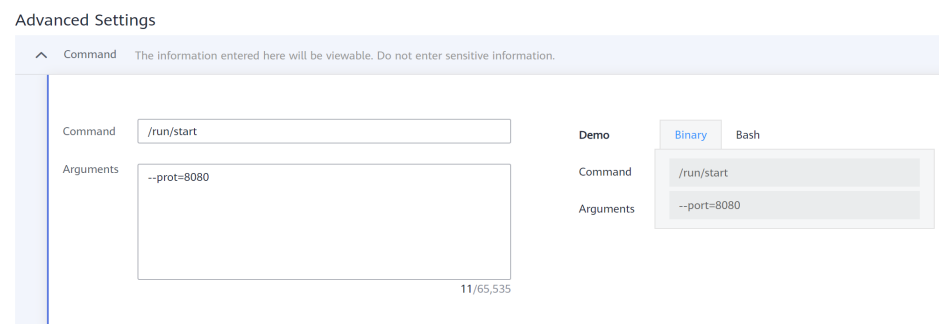
arguments provided during image creation, that is, **ENTRYPOINT** and **CMD** in the Dockerfile.

If the commands and arguments used to run a container are set during application creation, the default commands **ENTRYPOINT** and **CMD** are overwritten during image building. The rules are as follows:

**Table 4-15** Commands and arguments used to run a container

Image ENTRYPOINT	Image CMD	Command to Run a Container	Arguments to Run a Container	Command Executed
[touch]	[/root/test]	Not set	Not set	[touch /root/test]
[touch]	[/root/test]	[mkdir]	Not set	[mkdir]
[touch]	[/root/test]	Not set	[/opt/test]	[touch /opt/test]
[touch]	[/root/test]	[mkdir]	[/opt/test]	[mkdir /opt/test]

**Figure 4-94** Command



– **Command**

Enter an executable command, for example, **/run/start**.

If there are multiple commands, separate them with spaces. If the command contains a space, enclose the command in quotation marks ("").

**NOTE**

In the case of multiple commands, you are advised to run **/bin/sh** or other **shell** commands. Other commands are used as arguments.

– **Arguments**

Enter the argument that controls the container running command, for example, **--port=8080**.

If there are multiple arguments, separate them with line breaks.

• **Security Options**

- You can enable **Privileged Mode** to grant root permissions to the container for accessing host devices (such as GPUs and FPGAs).
- **RunAsUser Switch**  
By default, IEF runs the container as the user defined during image building.  
You can specify a user to run the container by turning this switch on, and entering the user ID (an integer ranging from 0 to 65534) in the text box displayed. If the OS of the image does not contain the specified user ID, the application fails to be started.
- **Environment Variables**  
An environment variable affects the way a running container will behave. Variables can be modified after workload deployment. Currently, environment variables can be manually added, imported from secrets or ConfigMaps, or referenced from **hostIP**.
  - **Added manually:** Customize a variable name and value.
  - **Added from Secret:** You can customize a variable name. The variable value is referenced from secret configuration data. For details on how to create a secret, see [Secrets](#).
  - **Added from ConfigMap:** You can customize a variable name. The variable value is referenced from ConfigMap configuration data. For details on how to create a ConfigMap, see [ConfigMaps](#).
  - **Variable reference:** The variable value is referenced from **hostIP**, that is, the IP address of an edge node.

 **NOTE**

IEF does not encrypt the environment variables you entered. If the environment variables you attempt to configure contain sensitive information, you need to encrypt them before entering them and also need to decrypt them when using them.

IEF does not provide any encryption and decryption tools. If you need to configure cypher text, choose your own encryption and decryption tools.

- **Data Storage**  
You can define a local volume and mount the local storage directory of the edge node to the container for persistent data storage.  
Currently, the following four types of local volumes are supported:
  - **hostPath:** used for mounting a host directory to the container. hostPath is a persistent volume. After an application is deleted, the data in hostPath still exists in the local disk directory of the edge node. If the application is re-created later, previously written data can still be read after the directory is mounted.  
You can mount the application log directory to the **var/IEF/app/log/{appName}** directory of the host. In the directory name, **{appName}** indicates the application name. The edge node will upload the .log and .trace files in the **/var/IEF/app/log/{appName}** directory to AOM.

**Figure 4-95** Log volume mounting

Local Volume Name	Type	Mount Directory	Permission	Operation
log	hostPath	/var/IEF/app/log/ngi	/var/log/nginx	Read/Write Delete

- **emptyDir**: a simple empty directory used for storing transient data. It can be created in hard disks or memory. emptyDir is an empty directory after being mounted. The application can read files from and write files into the directory. emptyDir has the same lifecycle as the application. If the application is deleted, the data in emptyDir is deleted along with it.
- **configMap**: a type of resources that store configuration details required by the application. For details on how to create a ConfigMap, see [ConfigMaps](#).
- **secret**: a type of resources that store sensitive data, such as authentication, certificate, and key details. For details on how to create a secret, see [Secrets](#).

---

**NOTICE**

- The container path cannot be a system directory, such as / or **/var/run**. Otherwise, an exception occurs. You are advised to mount the container to an empty directory. If the directory is not empty, ensure that the directory does not contain any files that affect container startup. Otherwise, the files will be replaced, making it impossible for the container to be properly started. As a result, the application creation will fail.
- If the container is mounted into a high-risk directory, you are advised to use an account with minimum permissions to start the container. Otherwise, high-risk files on the host machine may be damaged.

---

- **Health Check**

Health check regularly checks the status of containers or workloads.

- **Liveness Probe**: The system executes the probe to check if a container is still alive, and restarts the instance if the probe fails. Currently, the system probes a container by HTTP request or command and determines whether the container is alive based on the response from the container.
- **Readiness Probe**: The system invokes the probe to determine whether the instance is ready. If the instance is not ready, the system does not forward requests to the instance.

For details, see [Health Check Configuration](#).

**Step 6 Click Next.**

Containers can be accessed through a bridged network or a host network.

- **Bridged network**

The container uses an independent virtual network. A mapping between container and host ports needs to be configured to enable the external communication. After port mapping is configured, traffic destined for a host port is distributed to the mapping container port. For example, if container port 80 is mapped to host port 8080, the traffic destined for host port 8080 will be directed to container port 80.

 **NOTE**

For automatic port assigning during port mapping, enter a proper port range to avoid port conflicts.

You can select a host NIC to enable the port bound to this NIC to communicate with the container port. Note that port mapping does not support NICs with IPv6 addresses.

- **Host network**

The network of the host (edge node) is used. To be specific, the container and the host use the same IP address, and network isolation is not required between them.

**Step 7** Click **Next**. Confirm the specifications of the containerized application and click **Create**.

----End

## Status Description

A batch application deployment job can be in any of the following states.

- **Pending**: The job is waiting to be executed.
- **Running**: The job is being executed.
- **Successful**: All tasks in the job are executed successfully.
- **Partially successful**: Some tasks in the job are executed successfully.
- **Failed**: All tasks in the job failed.
- **Stopping**: The job is being stopped.
- **Stopped**: The job is stopped.
- **Update timed out**: The job is pended for more than 10 minutes or the job has not completed even after 10 minutes.

A job can be stopped during execution and resumed after being stopped.

If a job fails to be executed, partially succeeds, or times out, you can retry the job.

## 4.6.4 Upgrading Applications in Batches

 **NOTE**

The architecture of the container image to be used must be the same as that of the edge node on which a containerized application is to be deployed. For example, if the edge node uses x86, the container image must also use x86.

**Step 1** Log in to the IEF console, and click **Switch Instance** on the **Dashboard** page to select a platinum service instance.

**Step 2** In the navigation pane, choose **Batch Management** > **Application Upgrade Jobs**, and click **Create Application Upgrade Job** in the upper right corner of the displayed page.

**Step 3** Specify basic information about the application upgrade job.



**Figure 4-96** Upgrading applications in batches

The screenshot shows a web form for configuring an application upgrade job. It has three main sections:
 

- Name:** A text input field with the placeholder 'Enter a job name.'
- Description:** A larger text area with the placeholder 'Enter a job description.' and a character count '0/255' at the bottom right.
- Tags:** Two input fields labeled 'Tag key' and 'Tag value', with a note below them stating 'You can add 20 more tags.'

 Below the form is a blue-bordered box containing a note: 'Note: The upgrade configuration will be applied to all the selected applications.' At the bottom left of the form area, there is a label 'Upgrade Object' and a button labeled 'Select Application'.

- **Name:** name of the application upgrade job to be created.
- **Description:** description of the application upgrade job.
- **Tags:** tag of the application upgrade job.
- **Upgrade Object:** containerized application to be upgraded.

Click **Select Application**, and select applications to be upgraded.

You can also click **Search by Tag** in the upper right corner of the page, enter the tag key and value, and click **Search** to filter out the containerized applications with the specified tag. Then, select the containerized applications to be upgraded and click **OK**.

**Step 4** Click **Next** and configure container information. The configurations are the same as that in **Step 5**.

**Step 5** Click **Next** and configure access settings. The configurations are the same as that in **Step 6**.

**Step 6** Click **Next**. Confirm the specifications of the containerized application and click **Create**.

----End

## Status Description

A batch application upgrade job can be in any of the following states.

- **Pending:** The job is waiting to be executed.
- **Running:** The job is being executed.
- **Successful:** All tasks in the job are executed successfully.
- **Partially successful:** Some tasks in the job are executed successfully.
- **Failed:** All tasks in the job failed.
- **Stopping:** The job is being stopped.
- **Stopped:** The job is stopped.
- **Update timed out:** The job is pended for more than 10 minutes or the job has not completed even after 10 minutes.

A job can be stopped during execution and resumed after being stopped.

If a job fails to be executed, partially succeeds, or times out, you can retry the job.

## 4.7 Auditing

### 4.7.1 IEF Operations Supported by CTS

#### Scenario

Cloud Trace Service (CTS) is a log audit service provided by HUAWEI CLOUD and intended for cloud security. It allows you to collect, store, and query cloud resource operation records and use these records for security analysis, compliance auditing, resource tracking, and fault locating.

With CTS, you can record operations associated with IEF for future query, audit, and backtrack operations.

#### Key Operations Recorded by CTS

**Table 4-16** IEF operations that can be recorded by CTS

Operation	Resource Type	Trace Name
Registering an edge node	node	createEdgeNode
Updating an edge node	node	updateEdgeNode
Deleting an edge node	node	deleteEdgeNode
Creating an edge node group	group	createEdgeGroup
Updating an edge node group	group	updateEdgeGroup
Deleting an edge node group	group	deleteEdgeGroup
Creating a device template	deviceTemplate	createDeviceTemplate
Updating a device template	deviceTemplate	updateDeviceTemplate
Deleting a device template	deviceTemplate	deleteDeviceTemplate
Registering a device	device	createDevice
Updating a device	device	updateDevice
Deleting a device	device	deleteDevice

Operation	Resource Type	Trace Name
Deploying an Edge-Connector application	device	deployConnector
Updating device twins	device	updateDeviceTwin
Updating the device access configuration	device	updateDeviceAccessConfig
Creating an application template	application	createApplication
Updating an application template	application	updateApplication
Deleting an application template	application	deleteApplication
Creating an application template version	appVersion	createAppVersion
Updating an application template version	appVersion	updateAppVersion
Deleting an application template version	appVersion	deleteAppVersion
Creating a ConfigMap	configmap	createConfigMap
Updating a ConfigMap	configmap	updateConfigMap
Deleting a ConfigMap	configmap	deleteConfigMap
Creating a containerized application	deployment	createDeployment
Updating a containerized application	deployment	updateDeployment
Deleting a containerized application	deployment	deleteDeployment
Querying a list of containerized applications	deployment	getDeploymentList
Querying a containerized application	deployment	getDeployment
Creating an endpoint	endpoint	createEndpoint
Deleting an endpoint	endpoint	deleteEndpoint
Adding a node certificate	node	AddNodeCert
Deleting a node certificate	node	deleteNodeCert

Operation	Resource Type	Trace Name
Upgrading the firmware	nodeFirmware	UpgradeNodeFirmware
Querying a list of application instances	Pods	getPods
Querying an instance	Pod	getPod
Creating a node registration job	product	createProduct
Deleting a node registration job	product	deleteProduct
Creating a node upgrade job/Creating an application deployment job/Creating an application upgrade job	batchJob	createJob
Stopping a node upgrade job/Stopping an application deployment job/Stopping an application upgrade job	batchJob	pauseJob
Deleting a node upgrade job/Deleting an application deployment job/Deleting an application upgrade job	batchJob	deleteJob
Restoring a node upgrade job/Restoring an application deployment job/Restoring an application upgrade job	batchJob	restoreJob
Retrying a node upgrade job/Retrying an application deployment job/Retrying an application upgrade job	batchJob	retryJob
Querying quotas	quota	getQuota
Updating a quota	quota	updateQuota
Creating a rule	rule	createRule
Deleting a rule	rule	deleteRule
Updating a rule	rule	updateRule
Creating a secret	secret	createSecret

Operation	Resource Type	Trace Name
Updating a secret	secret	updateSecret
Deleting a secret	secret	deleteSecret
Filtering resources by tag	Tags	filterTags
Adding or deleting tags in batches	Tags	batchAddDeleteTags
Adding a tag	Tags	addTag
Deleting a tag	Tags	deleteTag
Binding encryption data	encryptdata	bindEncryptdata
Creating encryption data	encryptdata	createEncryptData
Deleting encryption data	encryptdata	deleteEncryptData
Unbinding encryption data	encryptdata	unbindEncryptdata
Updating encryption data	encryptdata	updateEncryptData
Creating a gateway	gateway	createGateway
Creating a virtual service	gateway	createVirtualService
Deleting a gateway	gateway	deleteGateway
Deleting a virtual service	gateway	deleteVirtualService
Updating a gateway	gateway	updateGateway
Updating a virtual service	gateway	updateVirtualService
Creating a system subscription	Systemevent	createSystemevent
Deleting a system subscription	Systemevent	DeleteSystemevent
Enabling a system subscription	Systemevent	startSystemevent
Disabling a system subscription	Systemevent	stopSystemevent

## 4.7.2 Viewing Tracing Logs

### Scenario

After you enable CTS, the system starts recording operations performed on IEF resources. CTS stores operation records generated within a week.

This section describes how to view the records on the CTS console.

### Procedure

**Step 1** Log in to the CTS console.

**Step 2** In the navigation pane, choose **Trace List**.

**Step 3** Set the filter criteria and click **Query**.


The following filters are available:

- **Trace Source, Resource Type, and Search By**

Select the desired filter criteria from the drop-down lists. Select **Management** for **Trace Type** and **IEF** for **Trace Source**.

In the preceding information:

- If you select **Resource ID** for **Search By**, you need to enter a resource ID. Only whole word match is supported.
- If you select **Resource name** for **Search By**, you need to select or enter a specific resource name.
- **Operator**: Select a specific operator from the drop-down list.
- **Trace Status**: Select one of **All trace statuses, Normal, Warning, and Incident**.
- **Time range**: You can select **Last 1 hour, Last 1 day, Last 1 week, or Customize**.

**Step 4** Click  on the left of a trace to expand its details.

**Step 5** Click **View Trace** in the upper right corner of the trace details area.

----End

## 4.8 Permissions Management

### 4.8.1 Creating a User and Granting Permissions

This section describes how to use **IAM** to implement fine-grained permissions control for your IEF resources. With IAM, you can:

Entrust a cloud account or cloud service to perform efficient O&M on your IEF resources.

If your account does not need individual IAM users, you may skip over this section.

This section describes the procedure for granting permissions (see **Figure 4-97**).

 NOTE

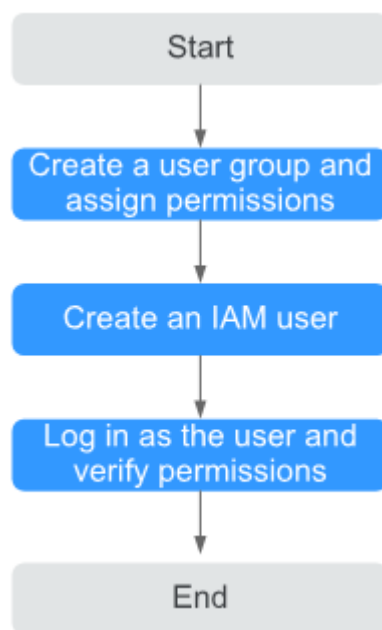
IEF supports only operation-level permission management and does not support resources or request conditions.

## Prerequisites

Learn about the permissions (see ) supported by IEF. For the system-defined policies of other services, see .

## Process Flow

**Figure 4-97** Process for granting IEF permissions



1. **Create a user group and assign permissions** to it.

Create a user group on the IAM console, and assign the **IEF ReadOnlyAccess** policy to the group. When assigning permissions to a user group, set **Scope** to **Region-specific projects**, and set parameters according to the following rules:

- To assign permissions in certain regions, select one or more specified projects, for example, **cn-north-4 [CN North-Beijing4]**. Note: If you select **All Projects** in this scenario, the authorization will not take effect.
- To assign permissions in all regions, select **All projects**.

**Figure 4-98** Assigning permissions in certain regions



**Figure 4-99** Assigning permissions in all regions



2. **Create an IAM user.**

Create a user on the IAM console and add the user to the group created in 1.

3. **Log in** and verify permissions.

Log in to the management console as the user you created and verify that the user has the assigned permissions.

- Choose **Intelligent EdgeFabric** from **Service List**. In the navigation pane, choose **Managed Resources > Edge Nodes**. On the displayed page, click **Register Edge Node** in the upper right corner. If you cannot register an edge node, the **IEF ReadOnlyAccess** policy has taken effect.
- Choose any other service from **Service List**. If a message appears indicating that you have insufficient permissions to access the service, the **IEF ReadOnlyAccess** policy has already taken effect.

## 4.8.2 IEF Custom Policies

IEF allows you to create custom policies.

You can create custom policies in either of the following ways:

- Visual editor: Select cloud services, actions, resources, and request conditions. This does not require knowledge of policy syntax.
- JSON: Edit JSON policies from scratch or based on an existing policy.

For details, see [Creating a Custom Policy](#). The following section contains examples of common IEF custom policies.

### Example Custom Policies

Allowing users to create and update applications and application templates

```
{
  "Version": "1.1",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ief:deployment:create",
        "ief:appVersion:update",
        "ief:deployment:update",
        "ief:application:create"
      ],
      "Condition": {
        "StringEquals": {
          "ief:AssumeUserName": [
            "test"
          ]
        }
      }
    }
  ],
  "Resource": [
    "ief:*:deployment:*",
    "ief:*:appVersion:*"
  ]
}
```



```
    "ief:*:application:*"  
  ]  
}  
]  
}
```

### 4.8.3 Entrustment Description

IEF works closely with multiple cloud services to use image, storage, data, and monitoring functions. When you log in to the IEF console for the first time, IEF automatically requests permissions to access those cloud services in the region where you run your applications. Specifically:

- **Application Operations Management (AOM)**  
IEF can collect performance metrics and logs (optional) of edge nodes and application containers through AOM, helping you monitor the performance of edge nodes and applications in real time and quickly identify risks.
- **Software Repository for Container (SWR)**  
IEF can manage and download self-defined container images through SWR, helping you deploy containerized applications on edge nodes.
- **Object Storage Service (OBS)**  
IEF can access created edge functions through OBS, helping you deploy and run functions on edge nodes and devices.
- **Data Ingestion Service (DIS)**  
IEF can send data of edge nodes and devices to your DIS streams.

After you agree to the entrustment, IEF automatically creates an agency in IAM to delegate other resource operation permissions in your account to Huawei Cloud IEF. For details, see [Account Delegation](#).

The agencies automatically created in IAM are:

- [ief\\_admin\\_trust](#)
- [ief\\_edge\\_trust](#)

#### ief\_admin\_trust

The `ief_admin_trust` agency has the Tenant Administrator permissions. Tenant Administrator has the administrator permissions on all cloud services except IAM. The permissions are used to call the cloud services that IEF depends on. The delegation takes effect only in the current region.

To use IEF in multiple regions, request for cloud resource permissions in each region. You can go to the IAM console, choose **Agencies**, and click **ief\_admin\_trust** to view the authorization records in each region.

#### NOTE

IEF may fail to run as expected if the Tenant Administrator permissions are not assigned. So, do not delete or modify the **ief\_admin\_trust** agency when using IEF.

## **ief\_edge\_trust**

The `ief_edge_trust` agency does not contain the Tenant Administrator system role. It only has the operation permissions of cloud service resources required by IEF and is used by edge nodes to report monitoring, alarms, and logs.

If the permissions of the `ief_edge_trust` agency are different from those expected by IEF, the console displays a message indicating that the permissions have changed and a re-authorization is required.

You may need to modify the permissions of the `ief_edge_trust` agency in the following scenarios:

- The permissions on which IEF components depend may change with versions. For example, if a new component depends on new permissions, IEF will update the expected permission list and you need to grant the required permissions to `ief_edge_trust`.
- When you manually modify the permissions of the `ief_edge_trust` agency, the permissions of the agency are different from those expected by IEF. In this case, a message is displayed, asking you to re-authorize the agency. If a re-authorization, the permissions you previously modified may become invalid.