



Technical concepts

Last updated: 03/03/2026

This content applies to the latest CD version of Cumulocity.

Specifications contained herein are subject to change and these changes will be reported in subsequent versions.

Copyright © 2026 Cumulocity GmbH.

The name Cumulocity GmbH and all Cumulocity GmbH product names are either trademarks or registered trademarks of Cumulocity GmbH and/or its subsidiaries and/or its affiliates and/or their licensors. Other company and product names mentioned herein may be trademarks of their respective owners.

This software may include portions of third-party products. Third-party terms are set out in a 3rd-party-licenses file linked to or included with each installation package.

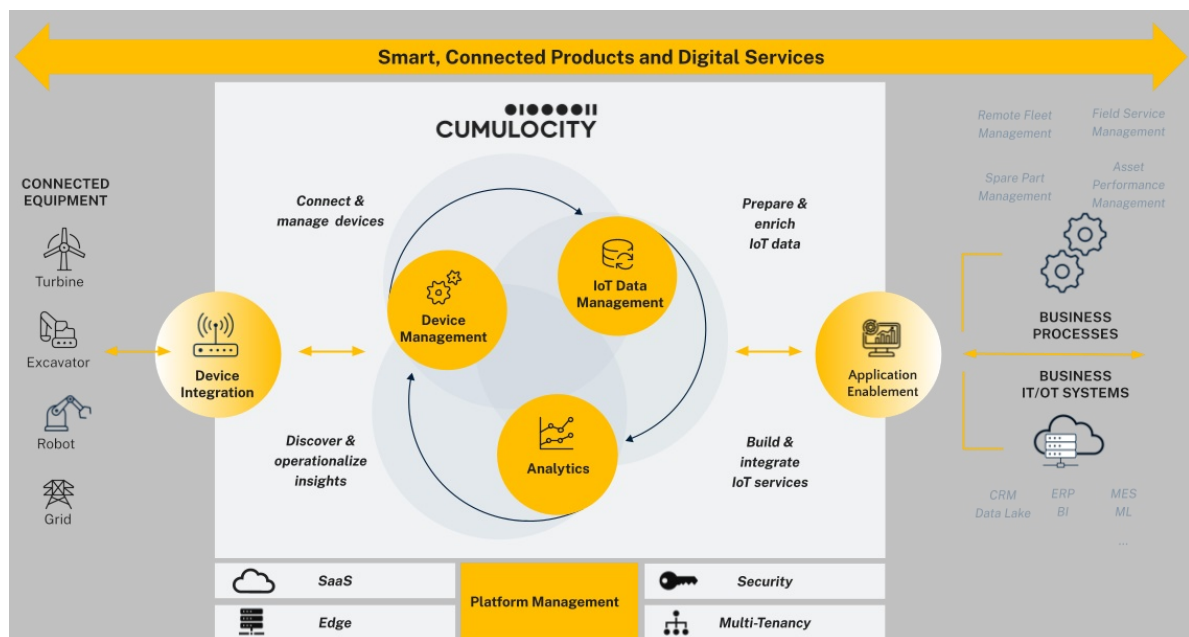
Table of Contents

Table of Contents	3
INTRODUCTION TO THE CUMULOCITY PLATFORM	4
DEVICE INTEGRATION	4
DEVICE MANAGEMENT	5
IOT DATA MANAGEMENT	5
APPLICATION ENABLEMENT	5
ANALYTICS	6
PLATFORM MANAGEMENT	6
Key Capabilities	6
Deployment options	7
CUMULOCITY'S DOMAIN MODEL	8
FRAGMENTS	8
Naming conventions of fragments	9
INVENTORY	10
MANAGED OBJECTS	10
OBJECT IDENTIFICATION	10
OBJECT HIERARCHIES	11
OBJECT LIFECYCLE	12
FURTHER INFORMATION	12
EVENTS	12
MEASUREMENTS	14
OPERATIONS	14
SENDING OPERATIONS TO DEVICES	15
DESIGNING OPERATIONS FOR RELIABILITY	16
SUMMARY	16
TENANT HIERARCHY	17
MULTI-TENANCY	17
HIERARCHY LEVELS	17
STANDARD TENANT	18
ENTERPRISE TENANT	18
MANAGEMENT TENANT	19
RBAC VERSUS MULTI-TENANCY APPROACH	19
INTRODUCTION	19
GENERAL SETUP	19
COMPARISON OF VARIOUS USE CASES	20
SECURITY ASPECTS	23
PHYSICAL SECURITY	23
NETWORK SECURITY	23
UNENCRYPTED COMMUNICATIONS	24
APPLICATION SECURITY	24
APPLICATION MANAGEMENT	24
ACCESS CONTROL	25
PERMISSIONS AND OWNERSHIP	25
LIMITING ACCESS TO MANAGED OBJECTS	26
MANAGING ROLES AND ASSIGNING PERMISSIONS	26
GLOBALLY ACCESSIBLE OBJECTS	26
MANAGEMENT SECURITY	26
SUMMARY	27

INTRODUCTION TO THE CUMULOCITY PLATFORM

Cumulocity is a robust, secure and scalable Internet of Things (IoT) platform designed to empower businesses in the following aspects:

- **Device integration:** Connect any type of device, from simple sensors to complex machinery, ensuring reliable and secure data transmission.
- **Device management:** Monitor, control, and manage all remotely connected equipment throughout its entire lifecycle, from deployment to retirement.
- **IoT data management:** Prepare, normalize, and enrich real-time and historical data, creating a unified view of your equipment.
- **Application enablement:** Visualize and analyze IoT data, integrate it with existing business systems, and rapidly build custom IoT applications and solutions to meet your business needs.
- **Analytics:** Automate, optimize, and remotely control equipment operations using intelligent analytics and real-time decision-making.
- **Platform management:** Manage configurations, security settings, and tenant structures in the cloud, or deploy the platform on your premises for full control.



This section introduces the core concepts and capabilities of Cumulocity, catering to IoT architects, automation engineers, industrial engineers, and developers.

DEVICE INTEGRATION

Cumulocity offers multiple methods for connecting devices:

- **Pre-integrated IoT gateways:** Choose from [certified devices](#) offered by Cumulocity certified partners for the easiest integration.
- **thin-edge.io:** [thin-edge.io](#) is recommended for custom device integration. Follow the [Getting started with thin-edge.io](#) tutorial for a hands-on example.
- **Direct integration:** Connect devices via [HTTP REST](#) or [MQTT](#) interfaces. MQTT interfaces are designed for minimal device-side logic, suitable for microcontroller-based devices.
- **LPWAN support:** Native [LWM2M support](#) and [LoRa Network Server integration](#) for Low-Power-Wide-Area-Network devices.

Once a device is connected to Cumulocity, it can start streaming equipment data into the platform. A common practice is to use a gateway device for data acquisition. These gateways are often connected to the different sensors and fieldbus devices of the equipment and responsible for collecting and forwarding relevant data to the IoT platform.

To facilitate the data integration, Cumulocity comes with an [OPC UA](#) integration and the [Cloud Fieldbus](#) technology. These provide configuration-driven ways to easily integrate OPC UA-enabled equipment and prominent fieldbus protocols like CAN, Profibus or Modbus.

In addition to this, many Cumulocity certified [partner devices](#) bring their own software stack with support for many other protocols.

INFO

The basic lifecycle for integrating devices into Cumulocity is described in [Interfacing devices](#).

DEVICE MANAGEMENT

Effectively managing devices is a critical yet often overlooked aspect of deploying IoT solutions. Without proper device management in place, it will become increasingly difficult to keep your devices healthy and up-to-date over time, leading to security risks, operational failures, and increased maintenance costs. This is especially true once you start scaling up your fleet of devices as well as adding new functionalities and thereby complexity to your deployment.

Connecting equipment to the internet requires secure and robust device management practices. Cumulocity greatly reduces the complexity and time required to manage a heterogeneous fleet of connected devices across its lifecycle through its comprehensive device management features:

- **Device onboarding:** Register and integrate any number of devices in just one go.
- **Over-the-air (OTA) firmware and software updates:** Keep your devices secure and up to date by efficiently rolling out the latest available software or firmware version.
- **Configuration management:** Easily change the settings of your devices by applying new configurations.
- **Connection and connectivity monitoring:** Quickly identify devices that stopped communicating and identify the underlying issue.
- **Remote troubleshooting:** Use a set of tools to resolve identified issues and minimize downtimes.
- **Device replacement:** Replace your physical devices without losing the data history.

IOT DATA MANAGEMENT

Cumulocity leverages a canonical yet extensible [data model](#) to represent data from different device types and protocols consistently. This approach decouples device integration from IoT applications.

To bring IoT sensor data into a meaningful context, Cumulocity offers the [Digital Twin Manager](#) (DTM). DTM enables users to model different [asset types](#) they are working with and their [properties](#). These models can then be used to create virtual representations of physical assets and their [hierarchical relationships](#). DTM then allows users to [link sensor data](#) directly to this asset hierarchy.

By modeling assets and their attributes, organizations can create a comprehensive view of their IoT data that is leveraged by all Cumulocity applications and can also be queried via [REST APIs](#).

Cumulocity also offers the optional [DataHub](#) application. [Cumulocity DataHub](#) allows for efficient [querying](#) of your IoT data using SQL via standard ODBC/JDBC interfaces for an efficient [integration](#) of your IoT data with external data lake stores. This capability is particularly valuable for organizations dealing with large volumes of data, as it allows for a seamless integration of the IoT data into existing Business Intelligence, Analytics and AI tooling and workflows.

APPLICATION ENABLEMENT

Cumulocity provides a comprehensive suite of tools and applications to make it easy to use the connected equipment to drive business outcomes.

At the core of this offering is the [Cockpit application](#) which offers a range of features that allow users to visualize fleet and equipment Key Performance Indicators (KPIs) through flexible [dashboarding](#), [create and manage reports](#) for data analysis and business insights and

efficiently [manage alarms](#) to ensure prompt response to critical events.

One of Cumulocity's core design principles is to allow for customization and extension.

For this, Cumulocity provides many self-service customization options such as the following:

1. **White-labeling:** All applications can be easily branded using the [branding manager](#), allowing organizations to maintain their visual identity.
2. **Custom dashboards:** Users can create tailored visualizations for their equipment and key performance indicators using the [dashboarding feature](#).
3. **Real-time analytics:** The Analytics Builder application enables users to define visual [real-time rules](#) for data processing and decision-making.
4. **Plugins and extensions:** A wide array of [plugins and extensions](#) is available to enhance platform functionality.

Cumulocity has also been designed with easy extensibility by developers in mind. This is enabled through well-documented open [APIs](#) exposing the complete platform functionality along with a [Command Line Interface \(CLI\)](#) for efficient development workflows and an active [developer community](#) providing a platform for knowledge sharing and problem-solving. All aspects of the platform are extensible:

1. **Device-side logic:** Developers can use the [thin-edge.io](#) framework for integrating device-side logic.
2. **UI applications:** The [Web SDK](#) allows developers to seamlessly extend any UI application with new functionality or leverage any Cumulocity UI component for your entirely own UI application.
3. **Backend services:** The [managed microservices hosting](#) together with the [Microservice SDK](#) makes it easy to develop, deploy and operate custom backend services.

For further reading, see the [Application enablement & solutions](#) section.

ANALYTICS

Cumulocity offers powerful analytics capabilities that enable businesses to extract valuable insights from their data as well as to operationalize derived insights using real-time rules.

For this, Cumulocity comes with real-time data streaming and processing capabilities, allowing businesses to process the incoming data and turn it into insights or actions. The predefined [smart rules](#) offer a wizard-driven approach to get started with data analytics, whereas the [Streaming Analytics application](#) offers flexibility to create even more rules, ranging from simple data transformations to more advanced business metric calculations like uptime or utilization rate, either for individual devices & assets or aggregated over a subset of the entire connected fleet. Real-time rules also allow for both triggering external workflows (such as the creation of a service ticket) as well as sending operations back to the equipment, for example, to optimize machine operations based on analytical insights.

Not all metrics and insights are solely relying on real-time incoming data. Metrics like "remaining useful life" of a device or asset include some kind of benchmark or learning based on historical data from that same or from similar devices or assets, and that is where AI and Machine Learning (ML) come in. The [Cumulocity DataHub](#) makes it easy for data analysts and data scientists to access the equipment data which is required to create and/or train any kind of AI model; the support for SQL via standard ODBC/JDBC interfaces allow for a seamless integration with the data science workbench or open source tooling of choice.

Bear in mind that the real value of AI/ML comes with the operationalization of the created models. You need to close the loop onto the incoming IoT data and switch from historical insight generation to real-time insight generation. Here again does our [Streaming Analytics application](#) come into play to support you in orchestrating the flow of data towards the deployed model and afterwards working with the model output to, for example, raise an alarm or trigger a workflow. For more information on machine learning capabilities, refer to the [Machine Learning introduction](#).

PLATFORM MANAGEMENT

Cumulocity provides a platform management solution that allows to efficiently control and secure IoT deployments, whether provided as Software-as-a-Service (SaaS) in the cloud or installed on-premises. Through a dedicated [Administration application](#), the platform allows administrators to manage key areas such as tenant and application management, user roles and permissions, and secure data sharing.

Key Capabilities

- **Multi-tenancy support:** With [Enterprise tenants](#), Cumulocity supports full [multi-tenancy](#), providing clear data separation and enabling organizations to securely manage multiple tenants from a single instance.
- **Application management:** Fully customize the look and feel of your tenants and applications using [white-labeling](#) and control, deploy, [subscribe](#), and [monitor](#) your custom [web applications](#), [extensions](#) and [microservices](#).
- **Advanced security and access control:** The platform offers fine-grained [Role-Based Access Control](#) (RBAC) to define precise permissions for devices, users, and services. Integration with existing Identity and Access Management (IAM) systems is supported through [Single Sign-On \(SSO\)](#) for streamlined access management, while [certificate-based authentication](#) ensures device-level security to prevent unauthorized access.
- **Data sharing and synchronization:** Using the [data broker](#), administrators can enable secure data exchange between different tenants and efficiently manage data sharing policies. This feature allows for safe and controlled device data sharing across different tenant environments.

Deployment options

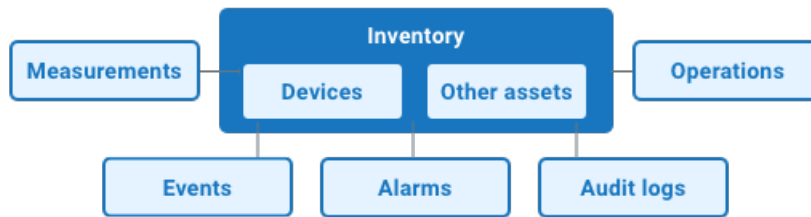
Cumulocity offers flexible deployment options to accommodate various business and regulatory requirements:

- **Cloud-based (SaaS) deployment:** Businesses can leverage the platform as a fully managed SaaS solution in the cloud, benefiting from seamless updates, scalability, and reduced infrastructure management overhead.
- **On-premises deployment with [Cumulocity Edge](#):** For environments requiring local installations, Cumulocity Edge provides the full capabilities of the platform on a single industrial PC or a locally managed Kubernetes. This option allows businesses to run the IoT platform independently from the cloud, making it ideal for environments with limited connectivity or stringent security requirements. The platform can operate in air-gapped networks.

With its consistent architecture, Cumulocity enables organizations to develop once and deploy seamlessly across both cloud and edge environments, ensuring unified management and consistent functionality. The [data broker](#) further supports this flexibility by enabling secure data synchronization between on-premises and cloud-based deployments, allowing for efficient data exchange while maintaining complete control over data security.

CUMULOCITY'S DOMAIN MODEL

The following image shows the relevant aspects of devices and assets in the Internet of Things:



- The **inventory** stores all master data related to devices, their configuration and their connections. It also contains all related assets (like vehicles, machines, buildings) and their structure.
- **Measurements** contain numerical data produced by sensors (like temperature readings) or calculated data based on information from devices (service availability of a device).
- **Events** contain other real-time information from the sensor network, such as the triggering of a door sensor. Events can also be **alarms**. The user or operator of the system must take action to resolve the alarm (like a power outage). In addition, security-related events are shown as **audit logs**.
- **Operations** relate to data that is sent to devices for execution or processing, such as switching a relay in a power meter or sending a credit to a vending machine.

One of the great innovations in Cumulocity is its standardized representation of common devices and sensors as well as concepts for flexibly extending and modifying this representation. By default, Cumulocity comes with detailed visualizations of sensors, smart meters, trackers and other devices. It has many options to fit in local customizations.

As a result, Internet of Things applications can be written independently from connected devices and underlying sensor networks, customized for specific cases in different web configurations or different devices from manufacturers.

FRAGMENTS

The key concept that allows for flexibly extending and modifying all objects in the Cumulocity domain model is called fragment.

To understand the fragment concept, imagine, for example, that you want to describe electric meters from different vendors. Depending on the make of the meter, one may have a relay and one may be able to measure a single phase or three phases. These characteristics are identified by storing fragments for each of them:

```

{
  "id": "47035",
  "type": "elstermetering_AS220",
  "lastUpdated": "2010-11-13T18:28:36.000Z",
  "c8y_Position": {
    "alt": 67,
    "lng": 6.15173,
    "lat": 51.211977
  },
  "c8y_ThreePhaseElectricitySensor": {},
  "c8y_Relay": {
    "state": "CLOSED"
  }
}

```

In this example, a fragment `c8y_ThreePhaseElectricitySensor` identifies a three phase electric meter. In addition, the device includes a relay, which can be used to turn the power supply on and off.

Using this approach, the modeling devices can make a difference between modeling elementary sensors and controls as fragments, and modeling the entire device as a combination of sensors, controls and possibly proprietary aspects of the device.

The approach also enables developing generic application components. For example, as soon as a managed object has a position fragment (`c8y_Position`), it can be placed on a map. As soon as it has a relay (`c8y_Relay`), it can be switched on and off using the

respective device control command as described below.

For more information about various Device Management functionalities and their associated fragments, see also [Fragment library](#).

While designing the data models, consider the following:

1. There is no size or length constraint for a single fragment, but there is a limitation for the overall JSON document size, which may not exceed 16MiB for a single managed object entry within the inventory collection. We recommend you to keep it below 1 MiB.
2. When designing asset hierarchies, use small groups with less than 1000 subassets. Each subitem in the asset hierarchy creates a reference record in the parent item. Therefore keep in mind the first recommendation regarding the JSON document size.
3. When you include arrays of elements within fragments, keep the length of such collections below 1k elements.
4. Each consecutive fragment added to the managed object at root level imposes a certain delay on querying such an item. If the performance of a query matters, it is recommended to nest custom fragments with information within a chosen single fragment effectively limiting the root fragments number. For example:

```
{
  "id": "47035",
  "type": "elstermetering_AS220",
  "lastUpdated": "2010-11-13T18:28:36.000Z",
  "c8y_ThreePhaseElectricitySensor": {},
  "c8y_DeviceMetrics": {
    "c8y_ConnectionMetrics": {
      "failures": 0,
      "successful": 1403,
      "total": 1403
    },
    "c8y_Alarms": {
      "requested": 100,
      "successful": 100
    },
    "c8y_Measurements": {
      "requested": 1303,
      "successful": 1303
    }
  },
  "c8y_DeviceMetricsConfiguration": {
    "deviceRepresentationUpdateIntervalCron": "0 22 * * *",
    "monitorApi": [
      "measurements",
      "alarms"
    ]
  }
}
```

Naming conventions of fragments

Fragments use a naming convention to avoid conflicts between different parties supplying fragment information, similar to Java or other programming languages.

In the example above, `c8y_Position` is a combination of “c8y” (a shorthand for “Cumulocity”), an underscore and “Position”.

❗ IMPORTANT

Names used for fragments must not contain whitespaces nor the special characters `. , * [] () @ $ / ' .`

Note that Cumulocity follows a document-oriented approach for storing data. All characteristics of an object can be inferred from the document with the object data itself. There is no explicit separate metadata model that must be configured and managed. However, applications can add own metadata and store values in the inventory additionally. For example, a vending application can maintain metadata about slot configurations of the diverse vending machine types in the inventory.

The following sections are a walk-through of these concepts and will describe the ideas behind it and give you examples which use Cumulocity's REST APIs as format. To use them with different programming languages, refer to the specified samples in the [Cumulocity OpenAPI Specification](#).

INVENTORY

MANAGED OBJECTS

The inventory stores devices and other assets relevant to your IoT solution. We refer to them as *managed objects*.

Managed objects can be “smart objects” such as smart electricity meters, home automation gateways and GPS devices. They can be assets you would like to monitor, such as rooms in which sensors are installed, or cars containing GPS devices.

The following JSON code shows a small example of a managed object in the inventory, in this case a simple switch.

```
{
  "id": "47635",
  "name": "Smart switch",
  "type": "ge_45609",
  "cBy_Relay": {
    "state": "OPEN"
  }
}
```

An example for another asset stored in the inventory could be a room in which a switch is installed (compare the `id` property of the above switch with the `managedObject` in the child asset reference below).

```
{
  "id": "59436",
  "type": "resortenergymgmt_Room",
  "name": "Sauna room",
  "childAssets": {
    "references": [
      {
        "managedObject": {
          "id": "47635"
        }
      }
    ]
  },
  "resortenergymgmt_RoomProperty": {
    "size": 56
  }
}
```

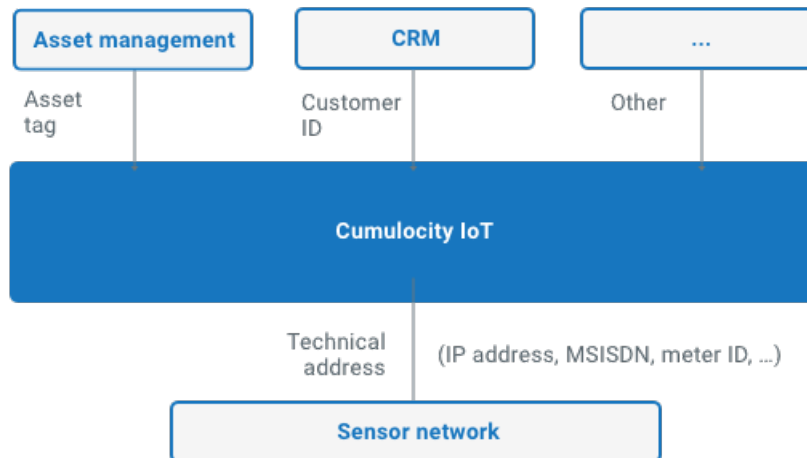
In general, each managed object consists of:

- A unique identifier that references the object.
- The name of the object.
- The most specific type of the managed object.
- A time stamp showing the last update.
- Fragments with specific meanings, for example, `cBy_IsDevice`.
- Any additional custom fragments.

OBJECT IDENTIFICATION

Each managed object in the inventory has an own, unique and global identifier that is automatically generated by Cumulocity when the object is created.

This identifier will always stay with the object regardless of network restructures or different hardware parts.



To shield applications from these numbers of identifiers, Cumulocity includes an identity service that registers all identifiers for one asset that are used outside of Cumulocity and map these to a single global identifier that is used by applications.

This service is used by agents (to register external identifiers) and by business processes involving reorganisations and changes of devices (to modify maps of external identifiers to global identifiers).

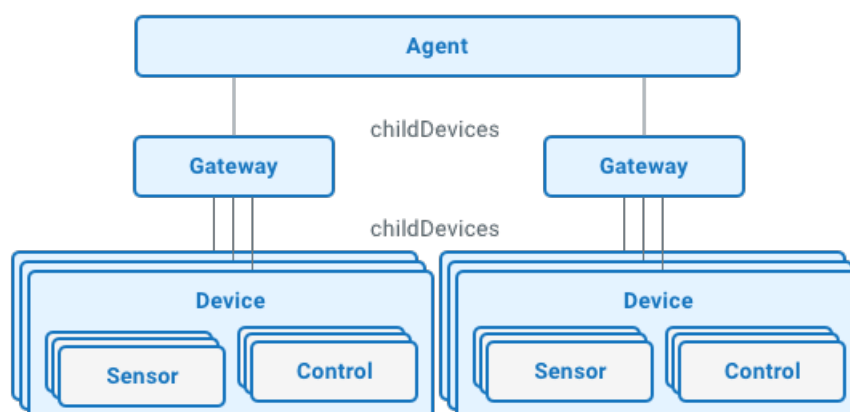
As an example, assume that a smart meter would be faulty and a new meter with another meter number and asset tag must be installed in a household. The routine business process for replacing faulty hardware can now just update the asset tag and meter ID associated with a customer in the identity service. Afterwards, both previously collected and new meter readings are related to the correct customer.

More information can be found in [Identity](#) in the Cumulocity OpenAPI Specification.

OBJECT HIERARCHIES

The inventory model supports two default hierarchies of objects: A communication hierarchy ("childDevices") and an asset hierarchy ("childAssets").

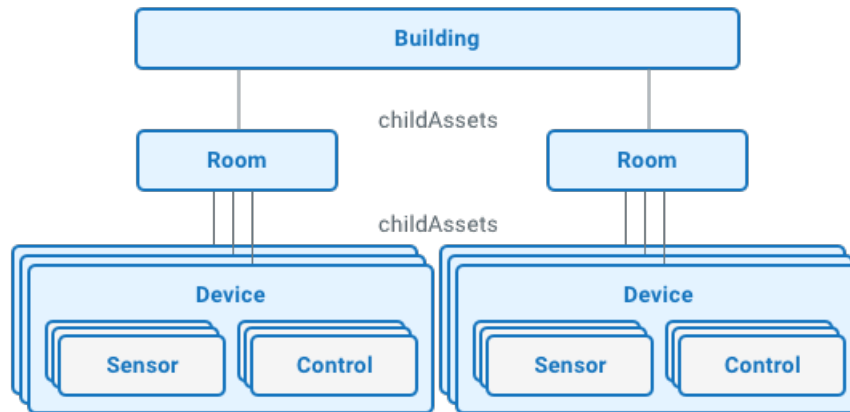
The communication hierarchy tracks how devices are linked to the IoT platform from a communication point of view. Agents connect the sensor network to Cumulocity. They often communicate through gateway devices or modems with the sensor network. The gateways, in reverse, connect to devices in the sensor network, which contain sensors and controls. This typical communication hierarchy is shown in the picture below.



The asset hierarchy structures the assets that are remotely supervised and controlled through the IoT devices.

An example asset hierarchy for building management could be buildings containing rooms. Buildings would be associated with gateways connecting the building to Cumulocity, while rooms would be associated with sensors and controls. This example hierarchy is shown in

the picture below.



Child objects in hierarchies

The two hierarchies above are explicitly supported by the [inventory interface](#) and client libraries, that provide methods for adding and removing children in hierarchies. The hierarchies themselves are constructed by client applications. The communication hierarchy is constructed by agents, the asset hierarchy is added by applications on top.

Note that the object hierarchies do not necessarily need to form a tree, the same asset can be a child of multiple parent assets. This enables applications to create additional, user-defined groups of objects such as working sets or virtual networks. Applications can, in addition, use fragments to define arbitrary alternative hierarchies.

OBJECT LIFECYCLE

The previously described identification and hierarchy mechanisms form a very flexible device lifecycle approach that can be adapted to most business processes. Initially, when a device is powered on for the first time, it is neither connected to the system nor linked to an asset. Linking a device to an agent in the communication hierarchy (possibly indirectly through a gateway) signals that the device is connected. Only connected devices can be remotely controlled. Linking a device to an asset using the asset hierarchy can be used to signal that the device has been physically installed.

Disconnecting and uninstalling a device does not necessarily indicate that the device was discarded or deactivated and should be deleted from the system. It can indicate instead that the device was returned to the warehouse and will be installed elsewhere later on. It depends on the particular business process whether data for the device should be kept or not. Physically deleting a device from the inventory implies that all data collected for that device is lost – this is probably desired only when completely cleaning up old data. To keep data for a device that has been discarded, identifier mappings can be removed from the identity service. Should a new device be installed in the same place as the old device, a new global identifier will be generated.

Addressing the device lifecycle properly is important when designing agents. An agent connecting to devices should not assume automatically that devices can be deleted from the inventory when they cannot be connected to. In the same way, an agent interfacing a CRM system should not assume that a device can be deleted when it has been removed from the CRM system.

FURTHER INFORMATION

More detailed information for working with the inventory can be found in the [Inventory API](#) in the Cumulocity OpenAPI Specification.

EVENTS

Events are used to pass real-time information through Cumulocity.

Events come in three types:

- A **base event** signals when something happens. An event can be triggered when a switch is switched on or off.

- An **alarm** signals an event that requires manual action, for example, when a meter has been tampered with or the temperature of a fridge increases above a particular threshold.
- An **audit log** stores events that are security-relevant and should be stored for auditing. For example, an audit log should be generated when a user logs into a gateway.

An event has one specific type (as specified in its naming convention), a time when the event occurred and a text to describe the event. An event refers to a source managed object in the inventory. This is an example of an event representation:

```
{
  "type": "c8y_LocationUpdate",
  "time": "2010-11-13T18:28:36.000Z",
  "text": "Location updated",
  "source": {
    "id": "47634"
  },
  "c8y_Position": {
    "alt": 67,
    "lng": 6.15173,
    "lat": 51.211977
  }
}
```

Any event can be extended in the same way as described for managed objects above. In this example, we not only signal that an object moved, we also include the new position of the object in the form of a `c8y_Position` fragment.

An audit log extends an event through:

- A username of the user that carried out the activity.
- An application that was used to carry out the activity.
- The actual activity.
- A severity.

This is an example of an audit record structure:

```
{
  "type": "c8y_SecurityEvent",
  "time": "2010-11-13T18:28:36.000Z",
  "text": "Gateway login failed",
  "source": {
    "id": "47633"
  },
  "user": "skywalker",
  "application": "Resort energy management",
  "activity": "login",
  "severity": "MINOR"
}
```

An alarm extends events through:

- A status showing whether the alarm is active or cleared.
- A time stamp when the alarm was last updated.
- A classification such as critical, major, minor, warning.
- A history of changes to the event in form of audit logs.

This is an example of an alarm that has been cleared:


```
{
  "count": 1,
  "creationTime": "2020-03-19T12:16:31.586Z",
  "id": "20200301",
  "severity": "MAJOR",
  "source": {
    "id": "251982",
    "name": "My tracking device"
  },
  "status": "CLEARED",
  "text": "No data received from the device within the required interval.",
  "time": "2020-03-19T00:00:00.000Z",
  "type": "c8y_UnavailabilityAlarm"
}
```

More detailed information can be found in [Events](#), [Alarms](#) and [Audits](#) in the Cumulocity OpenAPI Specification.

MEASUREMENTS

Measurements represent regularly acquired readings and statistics from sensors.

Measurements consist of a time when the measurement was taken, the unique identifiers of the source of the measurement, and a list of fragments. Here is an example of a measurement:

```
{
  "source": {
    "id": "251982"
  },
  "time": "2020-03-19T12:03:27.845Z",
  "type": "c8y_ElectricityMeasurement",
  "c8y_ThreePhaseElectricityMeasurement": {
    "A+": { "value": 435, "unit": "kWh" },
    "A-": { "value": 23, "unit": "kWh" },
    "P+": { "value": 657, "unit": "W" },
    "P-": { "value": 0, "unit": "W" },
    "A+:1": { "value": 123, "unit": "kWh" },
    "A-:1": { "value": 2, "unit": "kWh" }
  }
}
```

Similar to the inventory model, fragments are used to identify characteristics of particular devices. Each measurement fragment is an object that holds the actual measurements as properties, also known as series. The property name corresponds to the name of the measurement and includes two properties:

- value: A numeric value representing the individual measurement that is required for every series.
- unit: The unit associated with the measurement.

The example above represents a three-phase electricity meter that sends readings for the different electrical phases. A measurement fragment maps the names of the individual readings (for example, "A+" or "A-") to the actual numeric value and unit of the measurement.

In addition to value and unit, readings can hold various additional information that applications may require. However, detailed custom attributes are to be avoided in measurements. Instead, we recommend using the [events](#) domain model.

More detailed information can be found in [Measurements](#) in the Cumulocity OpenAPI Specification.

OPERATIONS

Devices can be remote-controlled and managed in Cumulocity.

Examples:

- Device control: Setting a switch which controls temperature.
- Device configuration: Setting up a charge table in a smart meter.
- Device maintenance: Requesting a gateway to download and install a new firmware.

In Cumulocity, these use cases are implemented by sending *operations* to a device.

The following snippet shows an operation for setting the state of the relay with the ID "42" to "OPEN":

```
{
  "deviceid": "42",
  "c8y_Relay": {
    "state": "OPEN"
  }
}
```

Just like other types of data, operations are also standardized to simplify application development (see below). For example, setting a switch should be the same for all switches regardless of their make.

Operations are modeled just like fragments in the inventory model (see above). The same extensibility concept applies. Random vendor-proprietary extensions to the standard operations are possible. These are not denied or modified by Cumulocity.

SENDING OPERATIONS TO DEVICES

Cumulocity delivers operations to devices over any network using a reliable queueing routine. This queueing routine respects the limitations and security requirements of IoT networks:

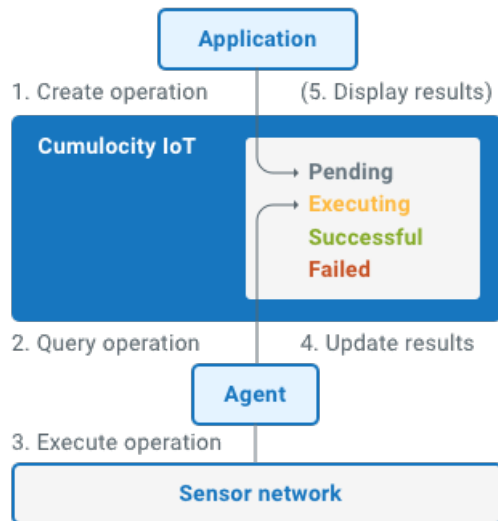
- Devices are often connected over unreliable, low-bandwidth links that may only occasionally be available. Devices may, for example, only dial up once in a day to the network to fetch commands for execution. Therefore Cumulocity communicates asynchronously with devices.
- Device protocols are often not designed for secure online communication. They may not pass NAT networks, firewalls and web proxies. They may not be secure enough for public exposure on the Internet. Cumulocity offers the possibility to connect these devices as HTTPS clients.
- It may not even be possible to reach a mobile device over the internet. Cumulocity uses push technology to send operations to devices.

To pass an operation from an application to a device, a process of several steps is required as illustrated in the image below. Assume that the user issues a remote control operation for a device (such as a device restart) from an application. The application creates the operation in Cumulocity (Step "1"). Cumulocity will queue the operation for execution and return control back to the application immediately.

At some point in time, the agent responsible for the device will request operations that are queued for the devices that it manages ("Step 2"). This will happen immediately through Cumulocity's push mechanism or at a regular or scheduled interval.

The agent will execute the operations on the devices that it manages (Step "3"), and will update Cumulocity with the results of the execution (Step "4"). The devices that the agent manages are direct or indirect children ("childDevices") of the agent.

Finally, the application can query the results of the operation (Step "5"). Audit records are generated both for the original request to run the device control operation and for the acknowledgement that the operation was actually run.



If there are communication issues while delivering an operation to a device, an alarm should be raised by the agent.

Sometimes there are delays between sending an operation to a device and retrieving a response. The system assumes a delivery unless an error is reported to maintain functionality.

DESIGNING OPERATIONS FOR RELIABILITY

Operations should always be idempotent. Idempotent means that no matter how often you run the operation, the outcome is always the same.

For example, an operation to set a switch to a certain state is idempotent. No matter how often the switch is set to “on”, it will be “on” afterwards. An operation to toggle a switch is not idempotent - the result depends on whether the operation was run an odd or an even number of times.

More details can be found in [Device control](#) in the Cumulocity OpenAPI Specification.

SUMMARY

Cumulocity provides a reference model for managing and controlling IoT systems, covering

- Central representation of IoT devices, networks and assets in the inventory.
- Configuration of devices.
- Reading of sensors.
- Manipulation of controls.
- Handling of real-time events.

This model is intended to be horizontal across device vendors. In addition it is also extensible to cover any needs of special features of various devices and applications.

TENANT HIERARCHY

Tenants are physically separated data spaces with a separate URL, with a specific set of users, a separate application management and no data sharing by default. Users in a single tenant share the same URL and the same data space.

For many scenarios it is sufficient to manage multiple parties within a single tenant. In this case, you [control the user access](#) by assigning appropriate roles - to specify which devices a user can see - and this way separate and protect multiple parties from each other. This approach corresponds to a [Standard tenant](#) in Cumulocity.

For other scenarios this approach might not be sufficient for various reasons and it might be relevant to manage a portfolio of tenants. This [multi-tenancy approach](#) is reflected by the concept of [Enterprise tenants](#) in Cumulocity.

MULTI-TENANCY

With the Enterprise tenant concept, Cumulocity supports full multi-tenancy. All data related to a tenant is stored in a dedicated data space. This includes user data, inventory, events, measurements, operations and alarms.

The Enterprise tenant can create subtenants that will then again function like Standard tenants in the platform and have their own tenant management.

This multi-tenancy approach has various advantages:

- **User and permission management**
Each tenant has full admin access to their user and permission management and can create its own roles.
- **Application management**
Each tenant can manage their applications separately and extend the platform by adding applications.
- **Usage statistics and billing**
Having separate tenants allows for cloud-based business models, which typically charge per API call and storage.

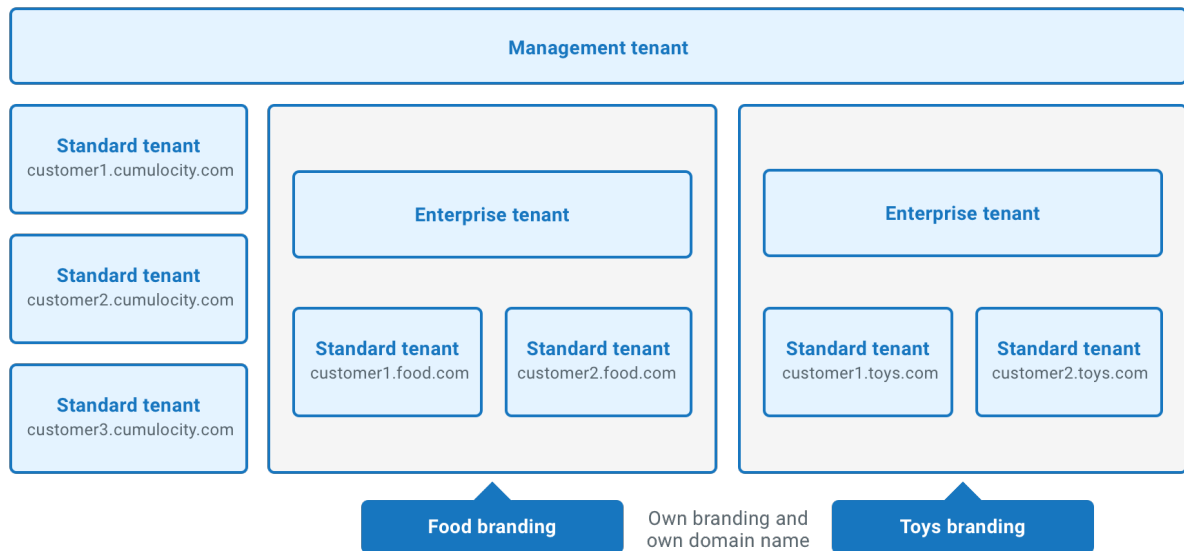
However, since every tenant has its dedicated database, only the IoT data of the tenant itself can be accessed within each tenant. While this has many advantages in terms of clear data separation it also means that, in order to share data between tenants it is required to copy the data over, which means that data is stored twice.

You can find a detailed discussion on the advantages and disadvantages of the role-based access control in a single tenant versus the multi-tenant approach in [RBAC versus multi-tenancy approach](#).

HIERARCHY LEVELS

The Cumulocity tenant concept builds a 3-level hierarchy, including the following levels from bottom to top:

- [Standard tenant](#)
- [Enterprise tenant](#)
- [Management tenant](#)



These three levels differ in their scope, particularly with regards to administration, see below.

INFO

Please refer to your contract for details on your individual subscriptions.

STANDARD TENANT

At the bottom of the hierarchy you can find single tenants which are represented by the concept of Standard tenants in Cumulocity.

A Standard tenant offers most of the device management and monitoring functionality of the Cumulocity platform, but has certain limitations when it comes to administrative aspects.

In a Standard tenant, multiple parties are reflected by separate users. Since all users in a tenant share the same URL and data space they can only be separated from each other by assigning access rights based on the [roles concept](#), that means, using roles you can give certain users only restricted visibility of the tenant (for example only the devices that belong to them).

Standard tenants, as direct subtenants of the Management tenant, provide fully standardized functionality.

Details on the administration of Standard tenants are described in [Standard tenant administration](#).

ENTERPRISE TENANT

An Enterprise tenant offers additional administrative functionality compared to a Standard tenant, the major difference being **multi-tenancy**.

Using an Enterprise tenant, you can

- create and manage subtenants
- manage the subscribed applications/features of the subtenants
- invoice subtenants based on usage statistics

See also [Multi-tenancy](#).

Moreover, an Enterprise tenant includes the following additional features:

- **Branding** - To configure an individual look & feel
- **Domain name** - To provide an individual domain name
- **Support user** - To support users of other tenants
- **User hierarchy** - To reflect entities with limited permissions to subsets of shared data

Enterprise tenants offer standardized functionality combined with some individual customization.

Details on the usage of this additional features and on the additional administration options of the Enterprise tenant can be found in [Enterprise tenant](#).

MANAGEMENT TENANT

The Management tenant builds the highest level of the Cumulocity tenant hierarchy.

Every Cumulocity deployment is delivered with a Management tenant. The Management tenant is used to administer all tenants within the same deployment on platform level and thus provides full control of the platform.

On the Cumulocity cloud instances, the Management tenant can only be accessed by the Cumulocity Operations team.

You will only have access to the Management tenant, when you setup your own Cumulocity instance either as a dedicated/hosted cloud deployment, an on-prem deployment, or the Cumulocity Edge offering.

For detailed information on the configuration options of a Cumulocity deployment on platform level, refer to the operations documentation, see [Additional resources](#).

RBAC VERSUS MULTI-TENANCY APPROACH

INTRODUCTION

When you think about offering your applications and services to your customers you must think at some point about how to structure your customers within the platform. Cumulocity can help you with that in two different ways.

The first is **role-based access control (RBAC)** that is part of every tenant in the Cumulocity platform and lets you granularly define the rights of each user. This can be used to give certain users (like a customer) only partial visibility of the tenant (only the devices that belong to them).

On top of that the Cumulocity platform in general is a **multi-tenant platform** which also gives you the ability to create your own subtenants, which function like any other tenant in the platform.

This will leave you with two options to organize your customers. You can either create a tenant for each of your customers or you can manage multiple customers within a single tenant and protect them from each other using RBAC.

In the following we will look at both approaches in more detail and run through some use cases explaining how to solve them in both ways. This should help you to decide which approach suits your business case better. Cumulocity is designed to manage tenants using the tenant hierarchy. As a result, some aspects that must be handled in a customer environment are more challenging when using Role Based Access Control. Using a combination of both approaches will provide you and your customers with the most flexible approach.

INFO

Starting with one approach and then switching to the other one will require some migration. It is easier to go from RBAC to multi-tenancy than vice versa.

GENERAL SETUP

Before going into detail with certain use cases we must clarify the general setup for each approach and the underlying concepts.

Role-Based Access Control (RBAC)

Handling everything in a single tenant usually starts by creating an asset hierarchy in which you create separate folders for your customers. Details of the asset hierarchy can be customized but at some point you will probably end up with one folder for each customer. Your customers will then only have access to their folders and will not be able to see anything outside of that folder.

Multi-tenancy

Creating a tenant for each customer will separate your customers at the tenant level. Your customers will get access to their tenants only and can work in this tenant the same way you do in yours (with whatever specific access you want to grant them). In the use cases we will assume that the customer is the admin of his tenant and therefore has full access.

INFO

The customer tenant is not different from your tenant unless you restrict your customer from certain functionality explicitly.

COMPARISON OF VARIOUS USE CASES

The following tasks should be covered by a platform solution:

- [Customer onboarding](#)
- [Device registration](#)
- [Access Rights](#)
- [User management](#)
- [Application management](#)
- [Invoicing and usage data](#)
- [Analytics](#)
- [Cumulocity DataHub](#)

The following sections discuss how these tasks are handled in both approaches.

Customer onboarding

Role Based Access Control

Adding a new customer is achieved by expanding your asset hierarchy and creating the respective user accounts for the customer with access to the newly created parts of the hierarchy.

Multi-tenancy

Adding a new customer is achieved by creating a new subtenant from your own tenant. During tenant creation you can automatically create the first user for the customer which will get admin permissions.

Comparison:

The creation of a new customer is equally simple. However, you must consider that in the multi-tenant approach you create a new empty tenant with nothing in it but the standard applications. You still might need to subscribe additional applications, create default dashboards, configure retention and others. These are already present in the RBAC approach as they are set up only once for everyone. On the other hand, this also means that you cannot have different setups for different customers, as certain settings (like retention) are configured at tenant level.

Device registration

Role Based Access Control

The common scenario in the RBAC setup is that the customer is not responsible for device registration, all devices are registered on the platform by the platform provider. However, it is still technically possible for a customer to register devices. The important detail in this case is that upon creating the registration entry the customer must specify the correct group to which the device belongs; otherwise the device would be created outside of any group and as customers can only see their groups they wouldn't be able to see the device.

Multi-tenancy

As customers have full access to their tenants they are free to register devices without any further limitations.

Comparison:

There is no technical limitation on who registers the device on the platform. However, care should be taken with the RBAC approach since customers can more easily make an incorrect configuration which registers the device without the customers being able to see it.

Access rights

Role Based Access Control

There are two types of roles in Cumulocity – global and inventory. Global roles are applied at the tenant level. In an RBAC approach you must use the inventory roles in order to have the correct level of separation. Apart from some global permissions (like "own user management") customer users will not be assigned any roles. Inventory roles must be created, or the default roles used, and then assigned to the user in combination with the assets the roles apply to. This must be done at least once for each customer.

Multi-tenancy

As the tenant is completely separated from all other customers you do not necessarily need to be involved in setting up the access rights of the customer. If customers are given administration rights for their tenants they can set up permissions on their own. It is not possible for customers to have any sight or knowledge of other customers.

Comparison:

In the RBAC approach, managing access is the most complicated part as a misconfiguration can potentially give customers access to data that they mustn't see, like other customers' data. The inventory roles allow you to granularly define access for only certain parts of data but they don't protect you from accidental misconfigurations. Another limitation here is that customers won't be able to create their own roles.

For security aspects on access control see [Access control](#).

User management

Role Based Access Control

Using the user hierarchy feature you can delegate the user management to the customers, so that they are able to create new users on their own. Using this feature they are only allowed to create users with the same roles that they have (or less). Therefore you must assign all roles that the customer needs in total to the first user so that he can delegate those.

Multi-tenancy

Customers have full admin access to the user management and can also define their own roles.

Comparison:

Having a separate tenant for each customer they will not be limited with respect to user management and can fully utilize all management functionality. In the RBAC approach you can delegate certain management functionality and the platform will make sure that users can never overstep their boundaries. However, certain functionality like creating roles will only be available to full user management admins.

Application management

Role Based Access Control

Application management can only be done by admins. Customers will still be able to grant their users access to available applications (of course only to those they can access themselves) but they won't be able to create own applications.

Multi-tenancy

Customers are free to add applications into their tenant as they see fit. The microservice hosting feature is optional and therefore must be granted to the tenant by the Management tenant. This does not apply for UI applications.

Comparison:

Application management is only available on tenant level. If you want to give customers the ability to extend the platform on their own you will need to go for a separate customer tenant. In the RBAC approach you need to take care of the application management but it is still possible to have different applications for different customers. This is easily doable for UI applications however if you add microservices you need to manage it via access rights as the microservice is generally available for the whole tenant.

Invoicing and usage data

Role Based Access Control

Multi-tenancy

The platform automatically collects usage data like API requests, storage space, number of users and devices. However, this is always on tenant level. Resolving which customer has how much devices is still doable via the API but you will not be able to separate storage space and API requests.

If your customers are subtenants of your own tenant you will be able to see and export the usage data for each tenant from your own tenant without requiring any access to your customers' data. You also have the ability to set limits for your customers, thus ensuring they stay under certain usage amounts.

Comparison:

Choosing the RBAC approach limits you in the options for your business model as it will be impossible to get accurate usage data for the customers. A license based business model (for example per device) is far more feasible in the RBAC setup. Dealing with a multi-tenant setup gives you the option to select typical Cloud-based business models where you charge per API call and storage.

Analytics

Role Based Access Control

All data is available in a single database and therefore you can easily apply analytics on the data either using the Apama streaming analytics engine or external tools. Applying analytics to just one customer can be a bit more challenging as you must split the devices based on the customer group they belong to.

Multi-tenancy

All data is split across multiple databases (one per tenant/customer) and you are not able to directly access all of them. You either need access to each tenant to extract the data or you are using features like data explorer to synchronize analytic-relevant data into a single tenant where you then can do your analytics.

Comparison:

If you are dealing with a single tenant it will be easier to do analytics across all devices of all customers but it might be more complicated to do separate analytics for just one customer. Having the data spread across multiple tenants will take additional effort to collect the data in one place for such use cases. However, it will ease deployment of custom analytics solutions per customer.

Cumulocity DataHub

Cumulocity DataHub currently does not support RBAC. Users who have access to DataHub on the tenant can offload any measurements, events, alarms, and inventory details into the same data lake regardless of the permission in use. Although it might be possible to restrict the offloading job to just data for a user this would require careful and manual configuration. DataHub currently only supports one data lake folder connection per tenant. Also note that only one user is created to access the data lake from analytical tools, therefore enforcing security on the data in the data lake is also currently not possible.

SECURITY ASPECTS

This section will show security concepts and aspects of Cumulocity, structured into physical security, network security, application security and access control. Finally, it shows how Cumulocity helps in managing the security of your IoT solution.

This section is especially intended for IT security staff and management staff. IT security expertise is required when running Cumulocity.

More information can be found in the security-related sections of the remaining documentation, like the [REST implementation](#) and [User API](#) description in the Cumulocity OpenAPI Specification. Permissions required for individual API calls are documented in the respective sections in the Cumulocity OpenAPI Specification.

PHYSICAL SECURITY

Physical security of IT systems prevents unauthorized physical access to servers, storage, and network devices.

Cumulocity Standard tenant accounts are hosted at Amazon Web Services (AWS). AWS has been certified according to [ISO 27001](#), [DSS and other standards](#). It features extensive physical security measures and is independently audited. Not all details are published for actual security reasons. Audit reports can be obtained directly at [AWS Compliance](#).

Our strategic hosting partners follow up to date concepts and concepts of data security.

In IoT solutions, physical security also includes unauthorized access to IoT devices, for example, to redirect or manipulate data from devices, read credentials from devices or change a device's configuration. We recommend you to review the physical security of the devices that you plan to use for your IoT solution and, for example, make **configuration ports unavailable** to unauthorized people or include tamper sensors as an additional security control within your own system.

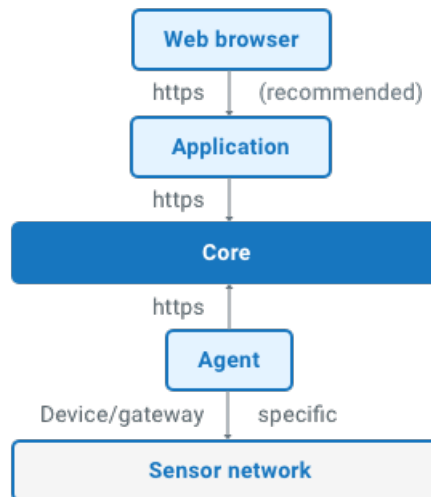
As the operator of the Cumulocity platform we do not control internal systems of our tenants. As a tenant you must follow a powerful and carefully considered security concept for your own system.

NETWORK SECURITY

Network security prevents unauthorized access to data transmitted over the network and tampering with or unauthorized modification of data. It also ensures that network services are available.

Cumulocity ensures that your data stays confidential and cannot be tampered with through an end-to-end implementation of [HTTPS](#) from devices to applications. It uses up-to-date encryption technology that has been independently rated "A" by [SSLlabs](#). Any communication with Cumulocity is subject to individual authentication and authorization.

This communication architecture is illustrated below. Inside the sensor networks and from the sensor networks to agents, device- and gateway-specific protocols may be in use (such as ZigBee or Modbus). Securing these is a device-specific matter. Agents communicate with the Cumulocity platform using HTTPS to send and receive data. Similarly, IoT applications use HTTPS for communication. If an IoT application exposes own interfaces towards web browsers, it is recommended that these use HTTPS. This way, the whole path from agents to the end user is secured.



As mentioned above, Cumulocity does not require any device that might expose ports or services on the internet. This is an important feature: it not only simplifies the connection of devices to Cumulocity, but also simplifies the safety backup of these devices drastically. When deploying an IoT solution, check other services that might make a device available on the internet or expose it, such as web-based device managers or SMS-based configuration options.

UNENCRYPTED COMMUNICATIONS

The Cumulocity platform allows old or low-power devices to connect to it over unencrypted protocols like HTTP since such devices are not capable of doing encryption or only provide weak encryption methods (for example TLSv1.0). Using old devices and unencrypted communications is discretionary and considering the risk that such communications are prone to attacks. For instance, an attacker suitably positioned to view a legitimate device's network traffic could record and monitor their interactions with the platform and obtain any information the device supplies. Note that using a mixture of encrypted and unencrypted communications is an ineffective defense against active attackers, because they can easily remove references to encrypted resources when these references are transmitted over an unencrypted connection.

Hence, it is recommended to use transport-level encryption (SSL/TLS) to protect all communications passing between the device and the platform. The Strict-Transport-Security HTTP header should be used to ensure that devices refuse to access the platform over an insecure connection.

APPLICATION SECURITY

Application security addresses security at the software level.

Cumulocity follows standard practices for application-level hardening as making sure that only properly upgraded operating systems and web servers are in use. A number of additional "best practices" are employed to make Cumulocity secure by design.

- Cumulocity supports multiple types of authentication methods, applying the best practices for each type. For more details on each type see [Authentication](#).
 - "Basic authentication" features sessionless REST APIs. This means that none of the popular "session stealing" techniques will work with Cumulocity.
 - "OAI-Secure" provides high security, using authorization tokens to prove the identity of the user.
 - "Single sign-on" redirect enables the use of an external authorization provider.
- Cumulocity does not use a SQL database for IoT data storage and is itself not based on a scripting language. This means that so-called "injection attacks" will not work with Cumulocity.
- As discussed above, devices are clients at Cumulocity and therefore popular attacks to devices will not work.
- Devices are individually connected with Cumulocity's device registration feature. This means that if a device is stolen or tampered with, it can be individually disconnected from Cumulocity.

APPLICATION MANAGEMENT

- Cumulocity uses a versatile permission model, which consists of application access control and REST API access control. This allows to precisely define the access rules for a user.

- In the Cumulocity platform, the standard applications are configured by default following the best security practices.

! IMPORTANT

Due to the very flexible nature of application management within Cumulocity, users can abuse certain security measures with sufficient permissions. For example, they can deploy a custom application with malicious code and make it available to the wide audience. Therefore, application management capabilities should be restricted to trusted and knowledgeable users.

i RELATED TOPICS

- [Platform administration > Standard tenant > Managing ecosystem](#) for details on managing applications in Cumulocity.
- [Platform administration > Standard tenant > Changing settings > Application settings](#) for details on managing application settings in Cumulocity.
- [Application enablement & solutions > Web SDK > Application configuration > Application options](#) for details on application configuration in Cumulocity.
- [Access control](#) for details on access control configuration.
- [Managing permissions and roles](#) for details on permission management.

ACCESS CONTROL

Cumulocity uses a standard authentication and authorization process based on realms, users, user groups, and authorities. A *realm* is a database of users and user groups, who follow the same authentication and authorization policy. A *user* is a person or an external system entitled to access protected resources inside Cumulocity.

Cumulocity creates a new realm for each tenant to store the users of that tenant. Realms provide an own namespace for usernames, allowing users to keep the names that they are familiar with from their own enterprise IT or other IT systems. There is no conflict between usernames: A user “smith” of one particular tenant is different from a user “smith” of another tenant. This username is valid for all Cumulocity applications that a tenant subscribes to.

Each new realm is automatically populated with an initial administrator user who can create further users and user groups (that is, global roles), and who can assign permissions to them. This enables an enterprise to manage users and their permissions on their own using the Administration application.

PERMISSIONS AND OWNERSHIP

The ability to execute certain functionality on the system depends on two concepts: Permissions and ownership.

Permissions define explicitly what functionality can be executed by a user.

Cumulocity distinguishes read permissions and administration permissions. Read permissions enable users to read data. Administration permissions enable users to create, update and delete data. Read and administration permissions are separately available for the different types of data in Cumulocity. For example, there are read permissions for inventory data, measurements, operations and so forth.

To manage permissions more easily, they are grouped into roles. When you assign one or more roles to a user, they gain the combined permissions of all these roles.

The following types of roles can be associated with users:

- **Global roles:** Contain permissions that apply to all data within a tenant.
- **Inventory roles:** Contain permissions that apply to groups of devices.

Objects in the inventory also have an owner associated with them. If you have created an object, you are the owner of it and can manage it without requiring any further permissions. Owners can always, regardless of their other permissions,

- Read, update and delete the inventory objects they own.
- Create, read, update and delete data associated with the objects they own.

For example, if you are the owner of a smart meter in the inventory, you can store meter readings for that smart meter even if you do not have any other measurement permissions.

The inventory also features a CREATE permission. A user having just the create permission can store new objects in the inventory, but can not read, modify or delete any other data. This is mainly relevant for devices. The CREATE permission also includes the possibility to link your object to another object as child device or child asset.

However, you cannot manage any devices or groups that you did not create yourself, unless you also have the UPDATE permission or an additional inventory role.

This concept helps to assign minimal permissions to devices.

LIMITING ACCESS TO MANAGED OBJECTS

Cumulocity allows you to set global permissions that are applicable to all managed objects, measurements, events and so forth. It also allows a limitation of permits

- to specific managed objects or a set of managed objects,
- to a single user or a group of users,
- to individual fragments.

MANAGING ROLES AND ASSIGNING PERMISSIONS

Global roles and inventory roles are created and managed in the **Roles** page of the Administration application in the UI.

A detailed description on available default roles and on creating and assigning global and inventory roles can be found in [Managing permissions and roles](#).

For details on permission management using the API refer to [the User API](#) in the Cumulocity OpenAPI Specification.

! IMPORTANT

In the Cumulocity API, each granular permission is identified by a unique “permission” string, which is prefixed with **ROLE_** (for example, **ROLE_ALARM_READ** , **ROLE_INVENTORY_ADMIN**). Therefore, permissions are frequently referred to as “roles” throughout the API as well as in the configuration files. For example, the microservice manifest includes a **requiredRoles** field which actually expects a permission string. See also the glossary for the usage of the terms “[permission](#)” and “[role](#)” in the Cumulocity context.

GLOBALLY ACCESSIBLE OBJECTS

To make a managed object, such as an inventory object, accessible to all users regardless of their individual permissions, add the **c8y_Global** fragment to the object. This applies only to inventory objects, not to alarms, events, etc., and the visibility does not extend to the object's children.

MANAGEMENT SECURITY

Whenever a security-relevant event occurs, it must be logged for potential auditing. Security-relevant events may happen both on application level as well as in the IoT network. A simple example of a security-relevant event on application level is a login to the application. An example of a security-relevant event on the network level is using a local software or local control on a device to manipulate the device.

To capture security-relevant events, Cumulocity offers an [auditing interface](#). This interface enables applications and agents to write audit logs, which are persistently stored and cannot be externally modified after being written. Cumulocity itself also writes own audit records related to login and device control operations.

To receive security-related reports about the Cumulocity platform, interested parties with a maintenance contract can subscribe to Early Warnings in the [Cumulocity Tech Community](#).

To report security incidents, please contact [product support](#).

SUMMARY

Cumulocity addresses security on various levels.

All business partners and service providers have recognized security certificates. Cumulocity also deals with network security aspects by individual authentication and authorization methods.

Connections from and to Cumulocity are established using HTTPS technology.

All tenants have full rights to add or terminate users and user groups. The tenant also assigns rights to agents and devices.