

TECHNICAL DOCUMENTATION

CoviDash

16th April 2020

IMPRINT

Your Contact Persons



Uwe Hartmann

E: Uwe.Hartmann@softwareag.com



Murat Bayram

E: Murat.Bayram@softwareag.com



Christoph Marsch

E: Christoph.Marsch@softwareag.com



Michael Tufar

E: Michael.Tufar@softwareag.com



Thorsten Illies

E: Thorsten.Illies@softwareag.com



Burkhard Hilchenbach

E: Burkhard.Hilchenbach@softwareag.com

INHALT

1	Management Summary	6
1.1	Content of this Document	7
2	General Solution Architecture and Central Capabilities	8
3	Solution Overview: Cumulocity (C8Y)	9
3.1	Solution Architecture: Standard Components	9
3.2	Device Management	10
3.3	Device Monitoring	11
3.3.1	Cockpit App	11
3.3.2	Device Management App	11
3.4	Microservice Runtime Environment: Create Own Applications	12
3.5	Long-term Data Storage with IoT Data Hub	14
4	Solution Overview: webMethods Integration Server (IS)	19
4.1	Integration Layer	19
4.1.1	Service development & orchestration	20
4.1.2	Service Enablement – Adapter	23
4.1.3	Service Enablement – Web Services	25
5	Cumulocity Domain Model	27
5.1	Inventory	28
5.1.1	Fragments	29
5.2	Events and Alarms	29
5.3	Measurements	31
6	Solution-Steps for Cumulocity (C8Y)	32
6.1	Tenant Creation	32
6.2	Cockpit Adaptation	33
6.2.1	Map	34
6.2.2	Asset Table	35
6.2.3	Alarm List	37
6.3	Smart Rules	38
6.4	Apama Analytics Builder	40
6.5	Bed Capacity Model	44
6.5.1	IVENA - Example	46

6.5.2	Rescuetrack – Example	46
7	Interface to Cumulocity	47
7.1	REST	47
7.2	Using the Interface and Postman Collection	47
7.3	Device SDK	48
7.4	Web SDK	49
8	Implementation in webMethods Integration Server (IS)	51

1 Management Summary

The German government initiated a Hackathon in order to bring digital solutions into a rapid stage where it can help people to overcome the crises of COVID-19. A Presales Team in DACH from Software AG decided to participate and use our reliable solutions to address a certain problem regarding the available bed capacities in Germany.

At the time of the Hackathon, there have been no central system in Germany for displaying free hospital capacities, in particular for free beds with COVID-19 capabilities. Hospitals in the federal states are still currently reporting vacancies to different data providers. At first to their regional provider, which aggregates these data to the respective regional rescue control centers. Secondly, meaning redundant double effort, to a newly created but standalone central “Deutsches Intensivregister”.

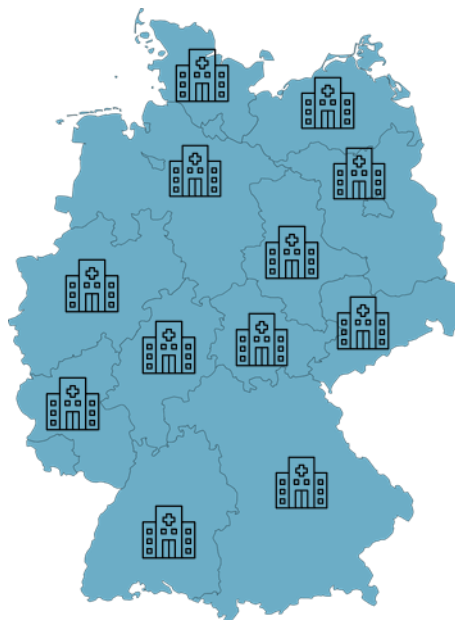


Figure 1: Germany and its federal structure.

Within the Hackathon the team created a consistent and future-proof IT- system on the basis of Cumulocity and webMethods with the name CoviDash to have a centralized and valuable view on the free vacancies in the hospitals based on a transparent and open data model.

In the first step for free beds to care for COVID-19 patients. Since Cumulocity’s data domain model allows various extensions it is the ambition to integrate further data (e.g. protective suits, breathing masks and respiratory equipment) to CoviDash in a later stage. Further motivation is, to establish a data standard for all federal states to make precise and valid statements and forecasts on the current and future COVID-19 pandemic disease.

In Germany are currently 4 service providers that already do have working systems with connections to the hospitals in each federal state.

rescuetrack and IVENA are two of the leading service providers which cover currently 10 federal states. Both providers have already given full support with regards to the technical deployment of their APIs.

Based on data from IVENA and rescuetrack, the first draft of a data model for the capacities of hospital beds was implemented in Cumulocity. Additionally, API integration connectors have been implemented within webMethods Integration Server (IS).

Since data sovereignty is the responsibility of each federal state, the individual ministries must approve the technical approval by IVENA and rescuetrack.

Due to the response from other countries the solution addresses certain problems in other countries as well. Since the structure of hospitals, states or governments might vary the solution need to be modified. With Cumulocity and webMethods Integration Server (IS) a very fast and valuable time to market is reasonable to achieve.

1.1 Content of this Document

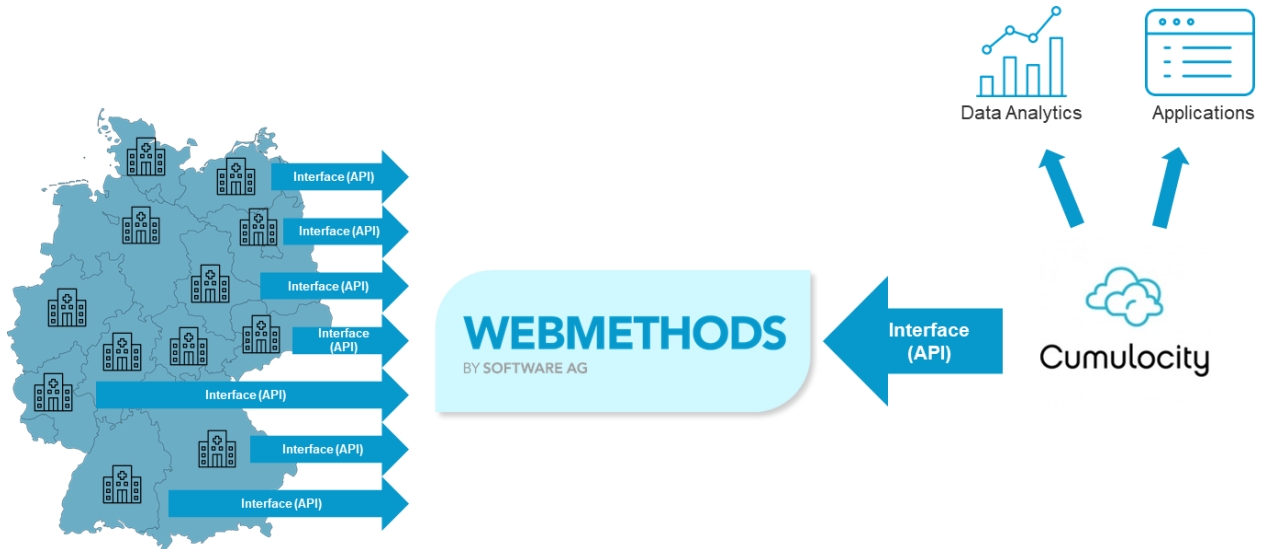
This document should give some instructions and architectural overview about the solution.

It therefore starts with standard capabilities of Cumulocity and webMethods Integration Server (IS), and how the different capabilities can be used. The Cumulocity Domain Model itself is explained in more detail since it is a very powerful method and allows various extensions in terms of a digital twin for a hospital if data provided.

For rapid start the basic key steps of the solution on Cumulocity side are shown as well as the advanced possibilities on how to connect devices, data to Cumulocity via REST and the SDK's.

2 General Solution Architecture and Central Capabilities

For the capabilities we defined during the Hackathon we propose the following solution architecture. It covers the requested capabilities in the most economical way.



The architecture proposed here is implementing the required capabilities as a public cloud solution that is highly secure, scalable and flexible in terms of deployment options (Azure, AWS, etc.) and third-party integration. Since the data might be classified as highly secure in some of the countries Cumulocity can run on an EDGE instance or as on premises installation as well. Since it's the same code basis the installation and look and feel will not vary. webMethods Integration Server (IS) can be installed on premise or in the cloud.

3 Solution Overview: Cumulocity (C8Y)

3.1 Solution Architecture: Standard Components

This section describes our **standard components** needed for achieving the basic requirements. We describe how Cumulocity IoT will provide connectivity, device management including identity management, remote access to devices, and device monitoring (including intelligent rules for advanced device monitoring).

The presented capabilities will enable a rapid go-to-market with industry-grade IoT services where Software AG in the background ensures scalability, security and operational excellence.

Cumulocity IoT is Software AG's carrier-grade, self-service IoT platform, which allows both business users and developers to rapidly create their IoT solutions and then extend them in scale and sophistication as requirements evolve. The multi-tenant platform is open with no vendor lock-in and can be fully distributed with deployments across cloud, on-premises, hybrid and edge. Native modular capabilities include device management, visualization, analytics, hybrid integration and full lifecycle application management. Each module is fully featured, with analytics covering condition monitoring, advanced streaming, predictive and timeseries. Cumulocity IoT is used by global brands like Siemens (MindSphere), Deutsche Telekom (Cloud of Things) and DELL.

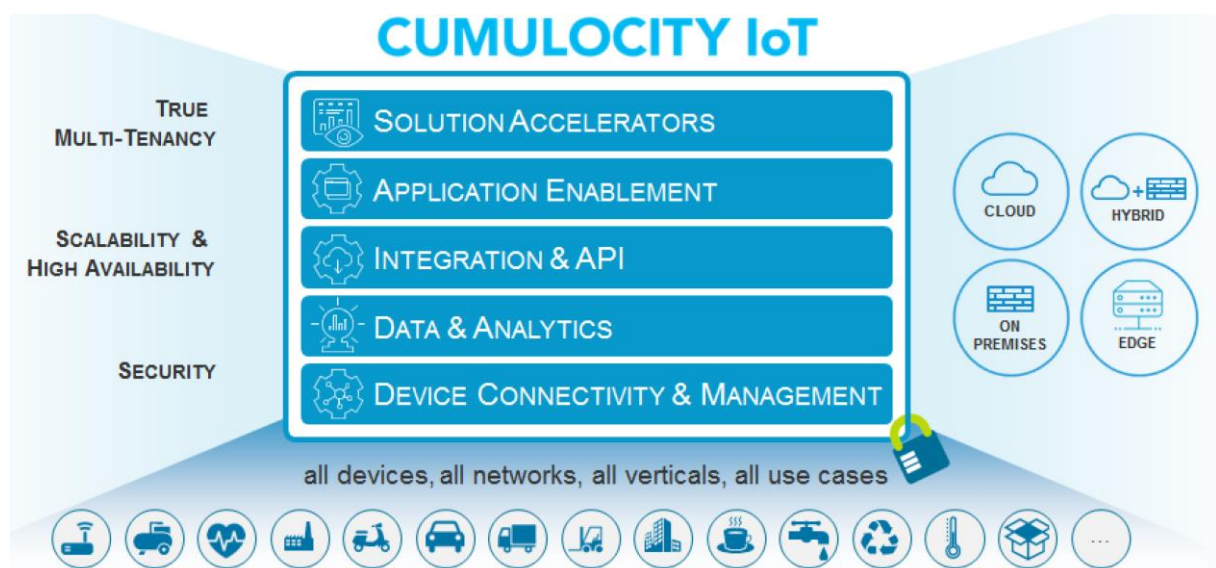


Figure 2: Key capabilities of the Cumulocity IoT platform.

3.2 Device Management

Cumulocity is device centric. Thus we modeled hospitals as devices. This can be seen as the first step of creating a digital twin of the hospital since Cumulocity's domain model allows implementation of assets due to the structure of frgements.

The Device Management application provides functionalities for managing and monitoring devices. Cumulocity IoT provides extensive device management including hardware and device agent information:

- Connection monitoring
- Centralized fault management and service level monitoring
- Configuration management
- Graphs of device statistics

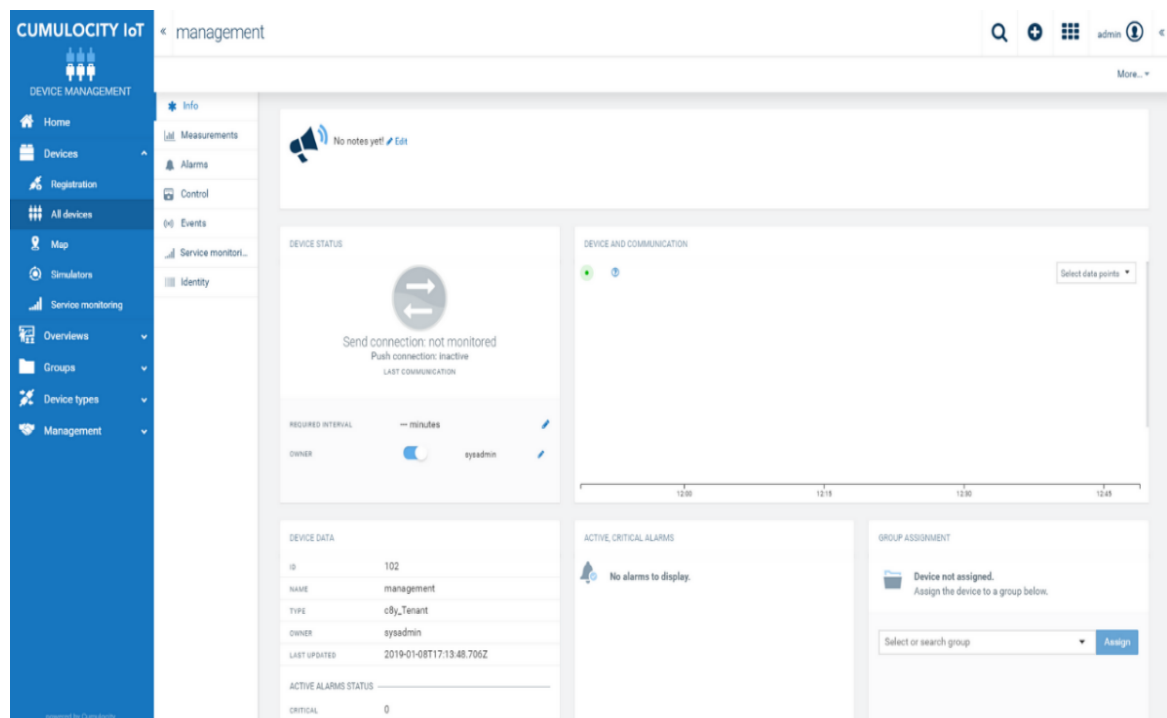


Figure 3: Overview information of a specific device.

To which level of detail device management is possible, obviously depends on the device.

3.3 Device Monitoring

Cumulocity IoT visualizes data centrally and graphically through its Angular-based web user interface.

The user interface automatically adapts itself to the web browser that is being used. For example, a mobile phone or tablet with limited screen size, it will change user interface controls to use less screen estate.

Generally, Cumulocity IoT comes with the following three standard applications:

Administration App

The Administration application enables account administrators to manage their users, roles, tenants, applications and business rules and lets them configure several settings for their account.

3.3.1 Cockpit App

The Cockpit application provides you with options to manage and monitor IoT assets and data from a business perspective.

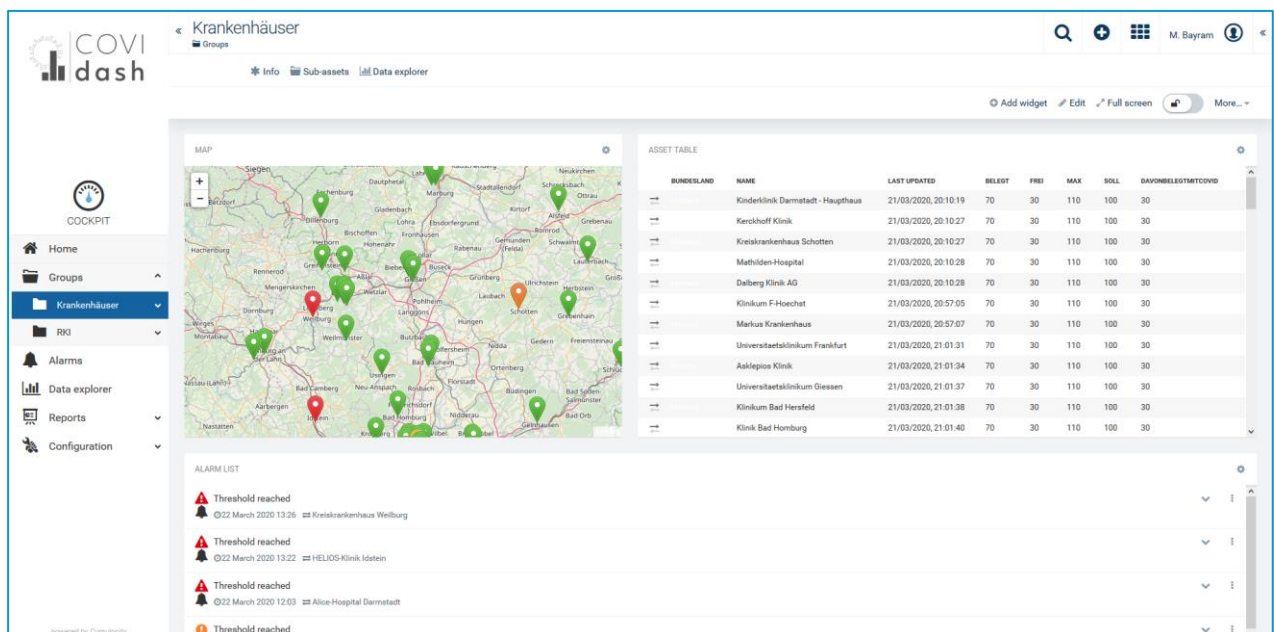


Figure 4: Example CoviDash demo dashboard created with the Cumulocity IoT Cockpit.

3.3.2 Device Management App

The Device Management application provides functionalities for managing and monitoring devices and enables you to control and troubleshoot devices remotely.

Beside those three standard applications, Cumulocity IoT provides a broad functionality to add your own applications to be used in your account. An application can be any combination of a complete, standalone user interface application, or a set of user interface plugins, or a set of statements in Cumulocity IoT Event Language. With Cumulocity IoT, users can publish any software to other users or customers (storing their data in other tenants). Those applications can be extensions of our base applications or entirely new user interfaces or new server-side business logic.

Applications are registered in Cumulocity IoT either as “own” applications or “market” applications. “Own” applications are only available to users of a tenant and are registered by the tenant’s administrator. Own applications are used, for example, during application development when you do not yet want to make a particular application version available for a wide audience. They are also used for functionality that is proprietary for an enterprise, for example, interactions with in-house IT systems. “Market” applications are available to all tenants of Cumulocity IoT. Subscribing a tenant to a market application makes this application available to the tenant and can be billed.

3.4 Microservice Runtime Environment: Create Own Applications

Microservices use standard REST APIs with full authentication and authorization to communicate with Cumulocity. They are, in most cases, multi-tenant, i.e. they need to be able to strictly separate tenants and connect to multiple tenants at the same time. Microservices may offer their own endpoints that can be used by Cumulocity and Cumulocity-based applications, e.g. for system integration purposes. Examples of such microservices are the Jasper Control Center integration and the SMS integration for sending SMS notifications to end users.

You can extend the Cumulocity platform with customer-specific functionality by employing microservices. For instance, you can develop integrations to third-party software or provide server-side business logic. A microservice-based architecture introduces change that is often well received by those developing modern applications, and solutions can be delivered much more quickly to those requesting flexible and scalable applications. Microservices bring significant benefits such as deployability, reliability, availability, scalability, modifiability and management.

Cumulocity microservices have the following properties:

- By default, they provide REST or Websocket APIs.
- Inbound REST and Websocket endpoints are secured by Cumulocity core built-in API gateway functionality.
- Requests from one microservice to the Cumulocity REST API can be executed by either using the original user account (of the inbound request) or by using a service user.
- Multi-tenant support.

The following management features are supported:

- Microservices can be registered to individual tenants and super-tenants (i.e. tenants with subtenants).
- Multi-tenant microservices can be subscribed to other tenants.

Technically, microservices are Docker containers hosted by Cumulocity and they follow specific conventions. They are typically accessed using Cumulocity REST API available under

/service/<microservice-name>.

Developers are not restricted to any programming language when developing a microservice for Cumulocity. However, a microservice must serve as an HTTP server working on port 80 and must be encapsulated in a Docker image.

Docker is a platform to develop, deploy and run applications with containers. An image is an executable package that includes everything needed to run an application (i.e. the code, a runtime, libraries, environment variables and configuration files). A container is a runtime instance of an image (i.e. what the image becomes in memory when executed).

A microservice is executed in a Docker container during runtime. The Docker container ensures that a microservice does not harm other microservices running in Cumulocity.

Kubernetes is the container orchestration engine for automating deployment, scaling and management of containerized applications. A Pod is the basic building block of Kubernetes and it represents a running process on your cluster. A Pod encapsulates an application container, storage resources, a unique network IP and options that govern how the container should run.

Docker is the most common container runtime used in a Kubernetes Pod. Moreover, Kubernetes is used to orchestrate Docker containers and it provides many enterprise-grade features for hosting Docker containers such as auto-scaling and load balancing. When Docker faces some issues, e.g. a Pod synchronization error, an alarm is created and can be seen in **Alarms** in the Cockpit application.

The following requirements towards Cumulocity microservices must be met:

- A microservice must be a (linux/amd64) Docker image run.
- The Docker image must be packaged as image.tar and must include a manifest file (cumulocity.json).
- A microservice must be stateless, i.e. it must contain only ephemeral state. The reason is that a microservice must be able to survive (random) restarts due to hardware reasons (server failure) and operations reasons (upgrade, migration).
- All persistent states must be stored at the Cumulocity platform via inventory, binary, tenant options and other APIs. Persistent volumes are not supported.
- A microservice cannot access the database directly and must use the Cumulocity API.
- A microservice must provide one inbound REST API. Additional inbound ports are not supported.
- A microservice can use multiple outbound ports.
- All log information must be sent to the standard output in order to be captured and persisted by the infrastructure.

3.5 Long-term Data Storage with IoT Data Hub

The Cumulocity IoT platform allows you to manage and monitor a variety of devices. The data emitted by these devices is stored in Cumulocity's Operational Store, with older data potentially being removed (based on data retention settings). In order to run an ad-hoc query against recent device data, Cumulocity offers a REST API.

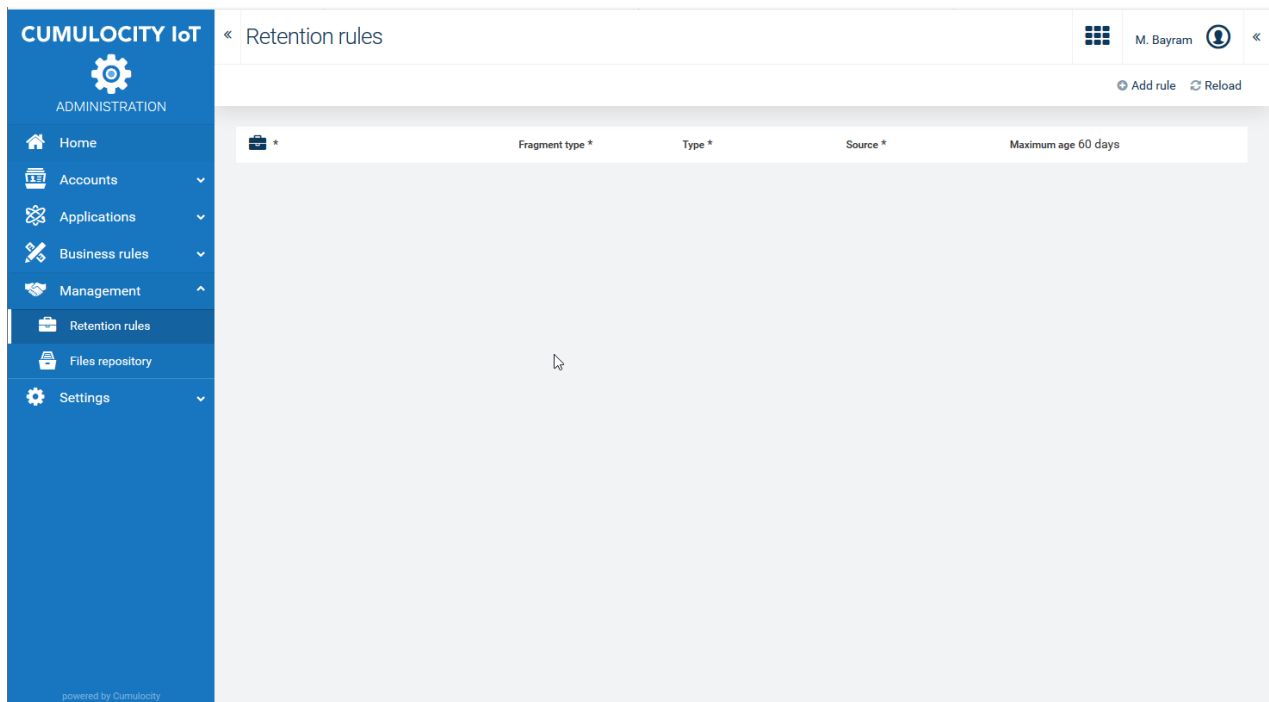


Figure 5: Retention administration for different data types.

In addition to this simple ad-hoc querying, various use cases require more sophisticated analytical querying over the device data, potentially covering long periods of time. Cumulocity IoT DataHub is the tool designed for this purpose.

With Cumulocity IoT DataHub, you can connect existing tools and applications to Cumulocity, such as:

- Business Intelligence/reporting tools (using ODBC, JDBC)
- Machine learning applications (mainly written in Python using ODBC)
- Arbitrary custom applications (using JDBC for Java applications, ODBC for .NET, Python, node.js, and others, or REST for Cumulocity web applications)

The main features of the Cumulocity IoT DataHub application are:

- It allows you to use scalable and inexpensive storage by providing an easy-to-use data pipeline that extracts data from Cumulocity's Operational Store to a **data lake** for long-term archival and efficient analytical querying.
- It offers an **SQL-based Query Interface** for querying the data lake and enables you to connect arbitrary applications that support ODBC, JDBC, or REST protocols.

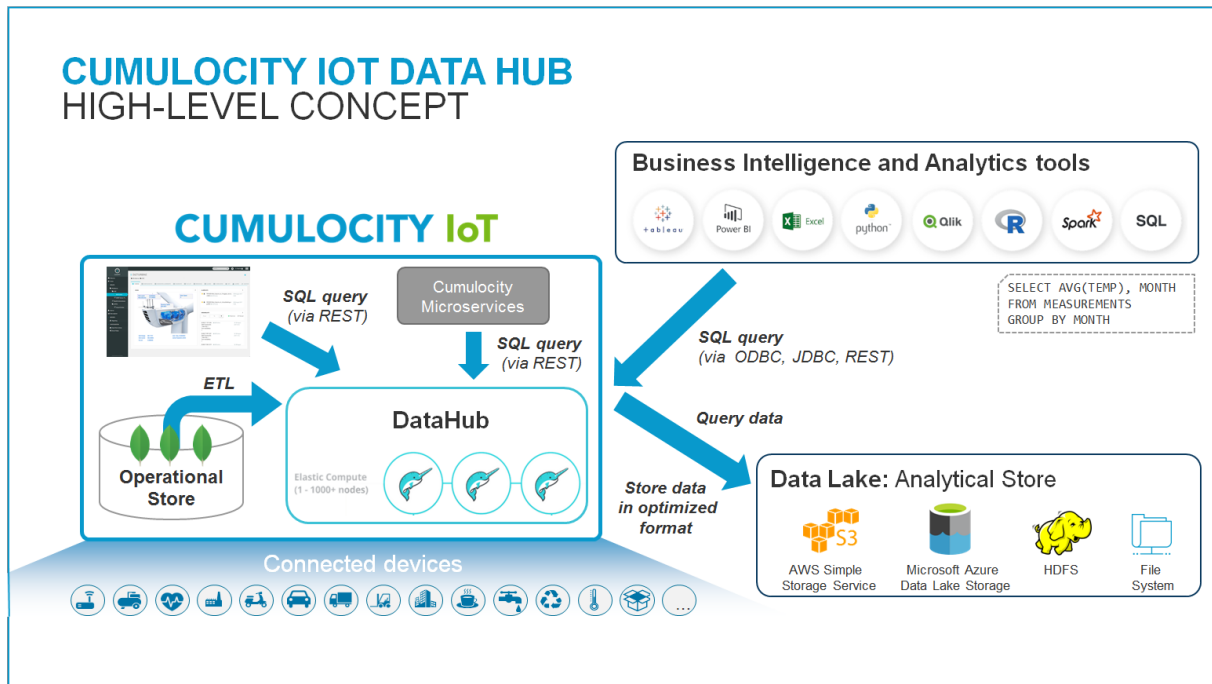


Figure 6: High-level concept of DataHub.

The central component of Cumulocity IoT DataHub is Dremio, a distributed SQL engine that is used for the two purposes mentioned above. It offers an SQL API which can be accessed via JDBC, ODBC, and REST. Dremio enables the creation of Extract-Transform-Load (ETL) pipelines that:

- Periodically extracts data from Cumulocity's Operational Store.
- Transforms the data into a relational format.
- Persists the data as Apache Parquet files in the configured data lake.

When a user submits an SQL query, the query runs against data in the data lake. Thus, Cumulocity's Operational Store is not accessed during query processing; the Operational Store is only accessed by the regular ETL process to extract data.

Offloading refers to moving data from Cumulocity's Operational Store to a data lake to:

- Provide the data in a flat and condensed format which can be leveraged for efficient SQL-based querying.
- Build a low-cost long-term archive of device data.
- Separate analytical workloads from operational workloads.

The starting point is one of the base Cumulocity collections, such as the measurements collection, that is to be offloaded into the data lake. Once an offloading pipeline for this collection has been configured and started, a couple of actions take place.

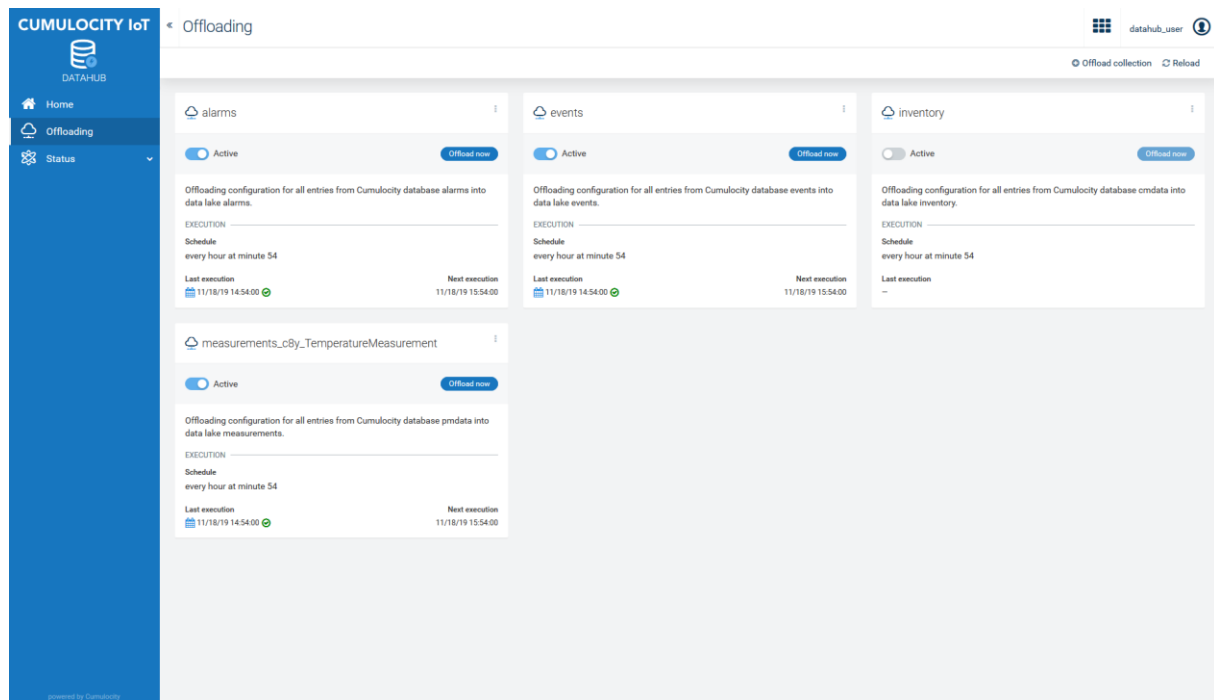


Figure 7: Overview about the offloading jobs within the DataHub application.

When an offloading job runs, the contents of the collection are offloaded. The document-based entities of Cumulocity's Operational Store are transformed into a relational format by flattening the entries and mapping them to relational rows.

Because of these extraction and transformation steps, the flattened data is stored in Parquet files in the data lake. Apache Parquet is a column-based storage format which allows for compression and efficient data fetching. For performance reasons, these Parquet files are managed in a folder structure based on a temporal hierarchy. The reason for the temporal hierarchy is that most analytical queries have a temporal background, e.g. compute the average oil pressure of last month. In order to ensure a compact layout of the Parquet files, DataHub Console also regularly runs a compaction algorithm over these files behind the scenes. When data is stored in a time-based hierarchical manner in the data lake, DataHub can efficiently prune partitions. In addition, queries can explicitly leverage the structure to increase query performance.

DataHub's scheduler runs the offloading pipeline in a periodic manner. The UI displays the execution schedule next to each configuration. Within each of these executions, newly arrived data is extracted from the Cumulocity collection and transformed and stored in the same way as described above. These incremental offloading tasks are designed to ensure a loss-free and duplicate-free offloading from the collection. For example, if one offloading execution fails, the next execution will automatically pick up the increments the failed one should have processed.

For each of the offloading pipelines, corresponding target tables are created in Dremio that point to the corresponding data folders in the data lake. When you run queries against the offloaded data, Dremio uses these target tables.

As described the Cumulocity IoT DataHub manages offloading pipelines which periodically extract data from Cumulocity's Operational Store, transform the data into a relational format and finally store it in a data lake. Instead of querying the Operational Store, you run your queries against the data lake. The distributed SQL engine Dremio provides the query interfaces to access the data lake.

Different standard interfaces exist for that purpose, namely JDBC, ODBC, and REST.

Id	creationTime	creationTimeOffset	creationTimeWithOffset	YEAR	MONTH	DAY	Time	TimeOffset	TimeWithOffset
2429664	2020-04-02 11:56:05.882	0	2020-04-02 11:56:05.882	2020	04	02	2020-04-02 11:56:04.000	120	20
2429684	2020-04-02 11:55:24.311	0	2020-04-02 11:55:24.311	2020	04	02	2020-04-02 11:55:24.000	120	20
2431389	2020-04-02 12:01:13.261	0	2020-04-02 12:01:13.261	2020	04	02	2020-04-02 12:01:12.000	120	20
2429578	2020-04-02 11:55:46.238	0	2020-04-02 11:55:46.238	2020	04	02	2020-04-02 11:55:46.000	120	20
2429540	2020-04-02 11:55:23.158	0	2020-04-02 11:55:23.158	2020	04	02	2020-04-02 11:55:23.000	120	20
2431816	2020-04-02 11:59:48.932	0	2020-04-02 11:59:48.932	2020	04	02	2020-04-02 11:59:48.000	120	20
2431261	2020-04-02 12:00:59.284	0	2020-04-02 12:00:59.284	2020	04	02	2020-04-02 12:00:59.000	120	20
2430724	2020-04-02 11:58:55.977	0	2020-04-02 11:58:55.977	2020	04	02	2020-04-02 11:58:55.000	120	20
2430642	2020-04-02 11:58:49.586	0	2020-04-02 11:58:49.586	2020	04	02	2020-04-02 11:58:49.000	120	20
2430591	2020-04-02 11:58:41.924	0	2020-04-02 11:58:41.924	2020	04	02	2020-04-02 11:58:41.000	120	20
2429768	2020-04-02 11:56:08.856	0	2020-04-02 11:56:08.856	2020	04	02	2020-04-02 11:55:59.000	120	20
2430584	2020-04-02 11:58:01.774	0	2020-04-02 11:58:01.774	2020	04	02	2020-04-02 11:58:01.000	120	20
2414248	2020-04-02 11:54:58.886	0	2020-04-02 11:54:58.886	2020	04	02	2020-04-02 11:54:49.000	120	20
2430319	2020-04-02 11:57:43.696	0	2020-04-02 11:57:43.696	2020	04	02	2020-04-02 11:57:43.000	120	20
2430184	2020-04-02 11:57:02.519	0	2020-04-02 11:57:02.519	2020	04	02	2020-04-02 11:57:02.000	120	20
2430696	2020-04-02 11:59:18.825	0	2020-04-02 11:59:18.825	2020	04	02	2020-04-02 11:59:09.000	120	20
2431357	2020-04-02 12:00:55.216	0	2020-04-02 12:00:55.216	2020	04	02	2020-04-02 12:00:54.000	120	20
2431386	2020-04-02 12:00:38.896	0	2020-04-02 12:00:38.896	2020	04	02	2020-04-02 12:00:29.000	120	20
2430224	2020-04-02 11:57:24.622	0	2020-04-02 11:57:24.622	2020	04	02	2020-04-02 11:57:24.000	120	20
2431245	2020-04-02 12:00:52.174	0	2020-04-02 12:00:52.174	2020	04	02	2020-04-02 12:00:51.000	120	20
2431325	2020-04-02 12:00:39.138	0	2020-04-02 12:00:39.138	2020	04	02	2020-04-02 12:00:38.000	120	20
2430431	2020-04-02 11:58:38.873	0	2020-04-02 11:58:38.873	2020	04	02	2020-04-02 11:58:38.000	120	20

Figure 8: Flattened data structure in an Amazon S3 data lake via Dremio UI.

When you have established a connection, you can run SQL queries against your tables in the data lake (to which new data is appended whenever the offloading pipeline has successfully run). The source you refer to in the query is defined by your account name and the target table you have specified in the offloading configuration.

For example, in this Covidash case you have defined an offloading configuration to write the alarms collection in an Amazon S3 Storage account containing a file system named covidash, then an example query would be:

```
Select * from t25691805DataLake.covidash.t25691805.measurement_bed_Capacities;
```

If you have a Java client, you can use JDBC to run SQL queries against the data lake. You can obtain the JDBC connection string from DataHub Console by clicking the JDBC icon in the Quick links section of the Home page.



Figure 9: Multiple connection types to query data. Native connector for PowerBi as an example (left) and a python script for extracting data via odbc connection string (right).

Due to the fact that ODBC and JDBC are common and well-known data base connectors there are already plenty of integrations within daily used software tools like e.g. Microsoft's Power BI. The connections and data extraction work out of the box and reports can be created based on long term storage data.

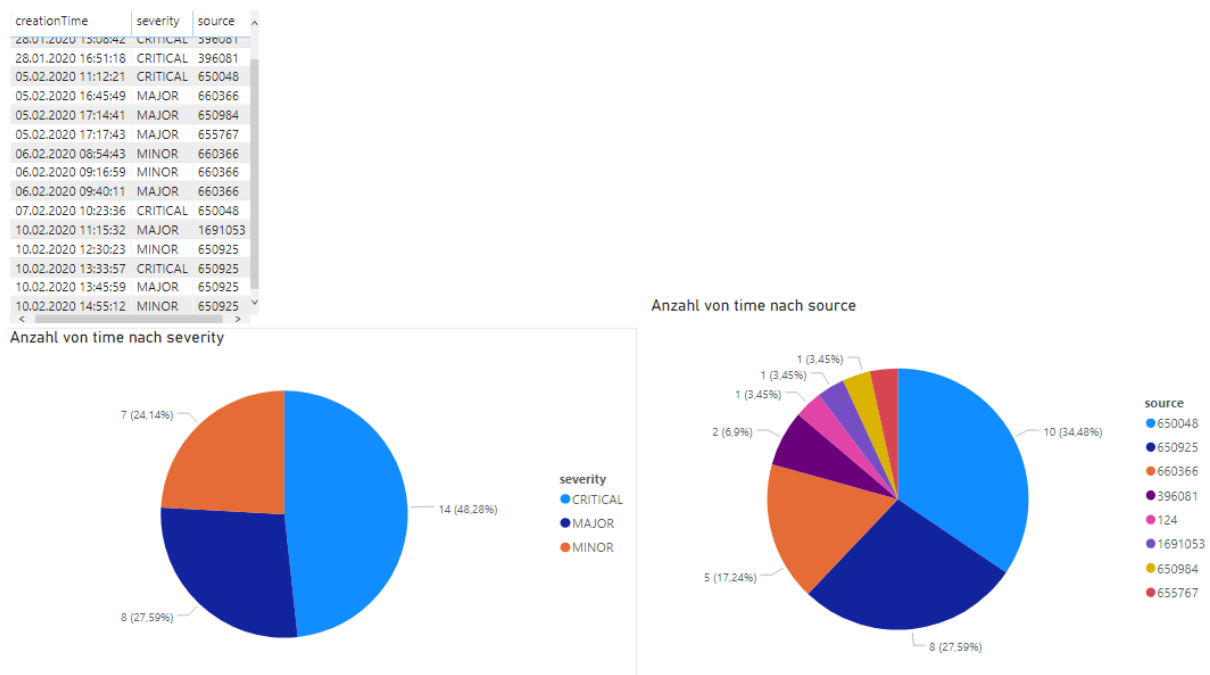


Figure 10: Power Bi Dashboard for the different alarm types and their sources for analysis purposes.

The same is true for the whole topic regarding data analysis or machine learning. Data scientist often use their own tools and frameworks such that forcing them to use one modeler tool would not work. For python, R, SPSS and others native connectors are provided in a way that data analysis can be archived in all common data analysis tools.

4 Solution Overview: webMethods Integration Server (IS)

4.1 Integration Layer

The capabilities of the Integration Layer are primarily delivered by the webMethods Integration Server (IS). The Integration Server is a robust platform supporting numerous standards for data transfer and web services, such as SOAP, REST, and JMS. Its architecture and capabilities allow building flexible and open service oriented as well as micro service architectures.

It is also the core runtime container for

- executing services of various flavors
- webMethods Monitor, for auditing of executed services
- the webMethods Adapter Runtime with its application and technology adapters
- webMethods Trading Networks, the suite's B2B solution
- webMethods ActiveTransfer, the suite's managed file transfer solution
- webMethods CloudStreams, the suite's cloud integration framework
- webMethods API Gateway, the suite's service and API mediation component

The fact that the Integration Server as a single runtime container is the main component for the execution of all integration, process automation and rules execution tasks is an obvious advantage. It significantly minimizes the development, testing, maintenance, administration and skill development effort.

The following figure depicts an overview of the high level architecture and capabilities of the Integration Server.

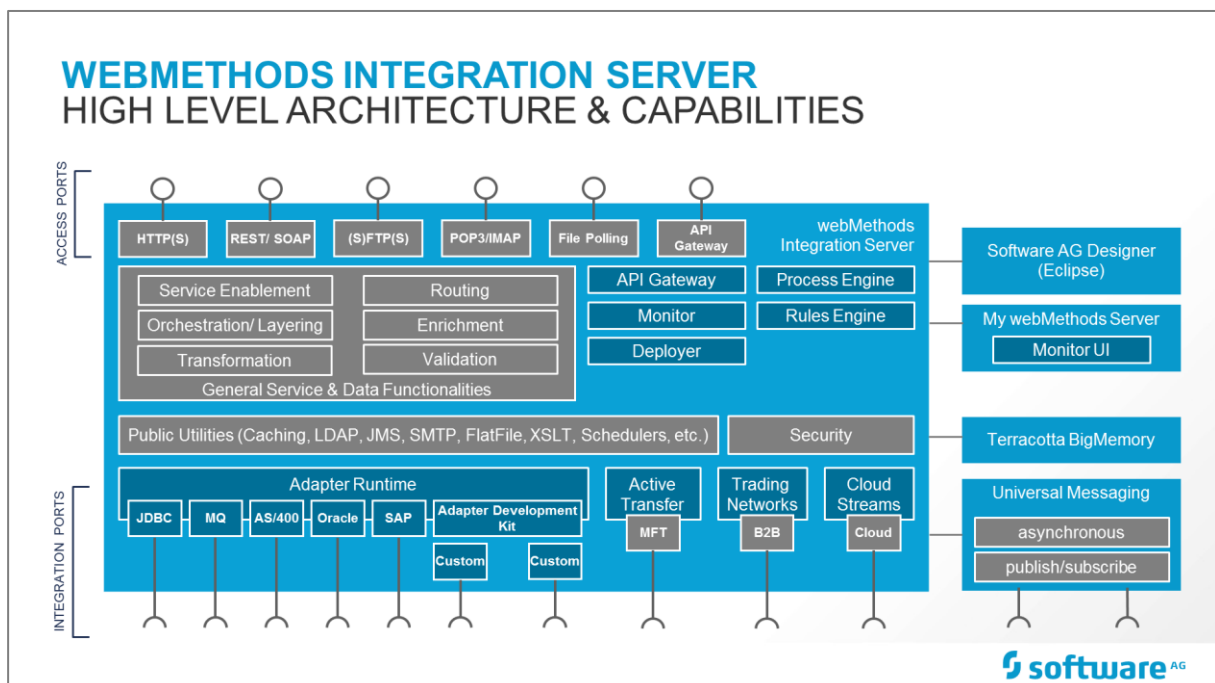


Figure 11: webMethods Integration Server – High level architecture & capabilities

In the following we'll describe the components and capabilities of the Integration Server in more detail.

4.1.1 Service development & orchestration

All development activities are generally performed in webMethods Designer, the unified Eclipse based development environment for the entire webMethods suite. You use it to develop integration services, automatically executable business processes models, graphical user interfaces for human workflows, business rules, mobile applications, streaming analytic applications and alike. Hence, there is no need for developers to switch between different tools for different development purposes. This makes working with the entire suite very convenient for any developer.

Within the scope of service and integration interface development as well as data mapping, it allows for point and click style, "codeless" development based on a graphical programming language called Flow. Flow is very easy to learn, as such delivers a steep learning curve, and enables users to dramatically speed up their integration development tasks. Of course, if desired, services can also be implemented in other languages like for example Java and C/ C++. However, experiences from other customer projects prove that the development time is significantly reduced when using Flow over Java or other languages for implementing integration services. Once individual services have been created, regardless of the implementation language being used, they can be easily composed to higher level service orchestrations by simply dragging and dropping them together and configuring their interaction in Flow.

The following figure shows a sample representation of a service implemented using the graphical development language Flow with data mapping between different data objects.

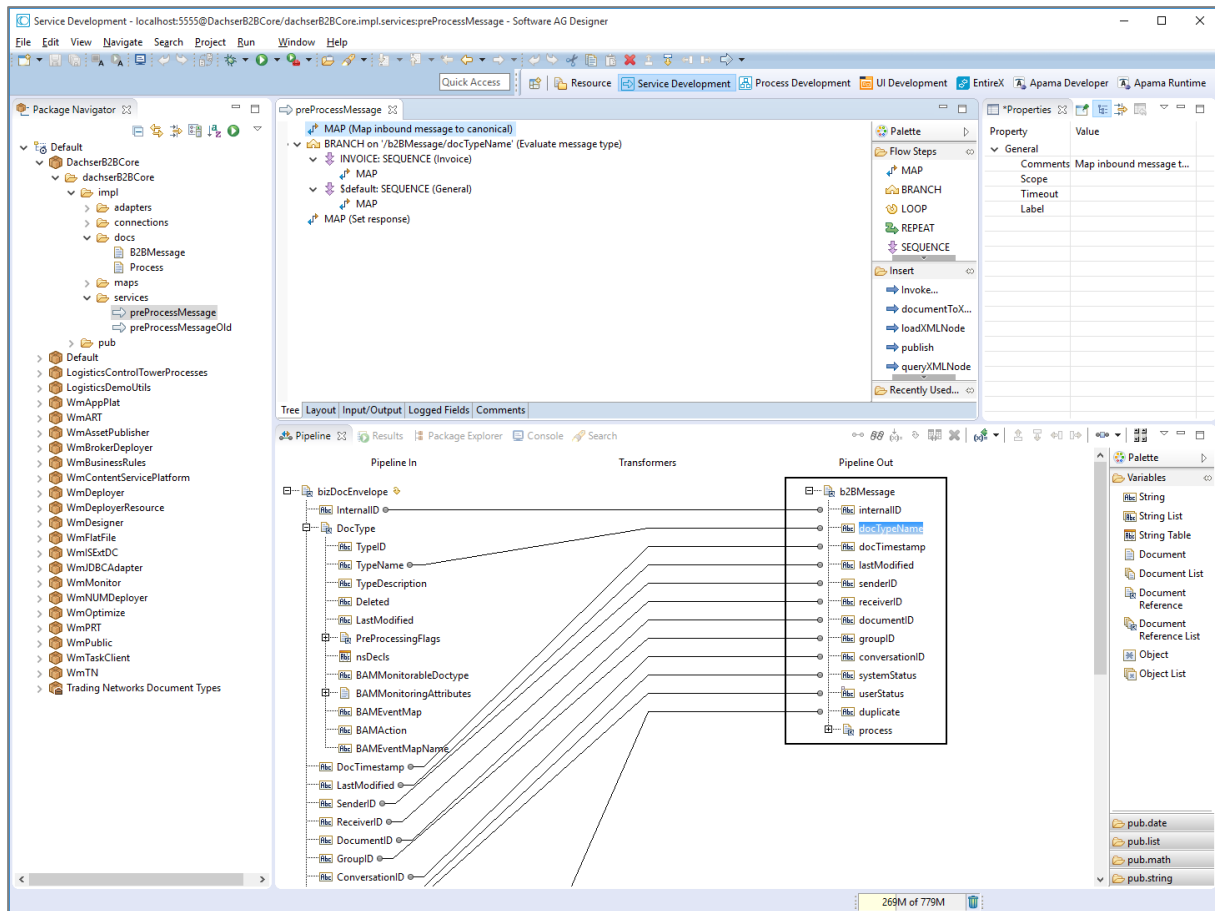


Figure 6: Fast and simple service orchestration & data mapping using the graphical development language Flow

Part of service development activities are typically also data and protocol transformations, data enrichment, data validations, security enforcement and scheduled service configuration. All of these functionalities can be easily implemented on Integration Server.

Integration Server comes with a large library of built-in services. Amongst many others functionalities, these services allow performing for example XSLT-based data transformations, schema-based data validations, integration with various messaging providers via JMS, in-memory cache interaction and digital encryption and signature enforcement.

A common requirement is also the processing of flat files in various formats. Using flat file dictionaries and schemas as well as the built-in flat file conversion services you can read, process, and create many different flat file data structure types, such as delimited, fixed length, and variable length. As especially B2B integration scenarios are often still file based, these rich flat file processing capabilities are particularly helpful.

Overall the service library comprises over 600 built-in services to simplify service development on Integration Server.

Additionally, you can implement arbitrary data structures. On the one hand side such data structures can be defined from scratch on the Integration Server using a graphical data editor. On the other hand side data structures can also be generated by importing already existing XML files, XML schemas, DTD files, JSON files, Adobe Lifecycle templates, Microsoft InfoPath forms, and SAP IDOCs.

In order to quickly find and resolve potential defects in the implemented code, Software AG Designer also provides debugging capabilities for Flow services as well as a dedicated Debug perspective. The features are similar to other programming languages. Amongst other things you can

- execute services step by step
- examine the service payload at each individual step
- modify the service payload at each individual step
- step over or into individual steps including nested child services
- set breakpoints to suspend execution at a predefined step

The following figure shows a debug session as an example.

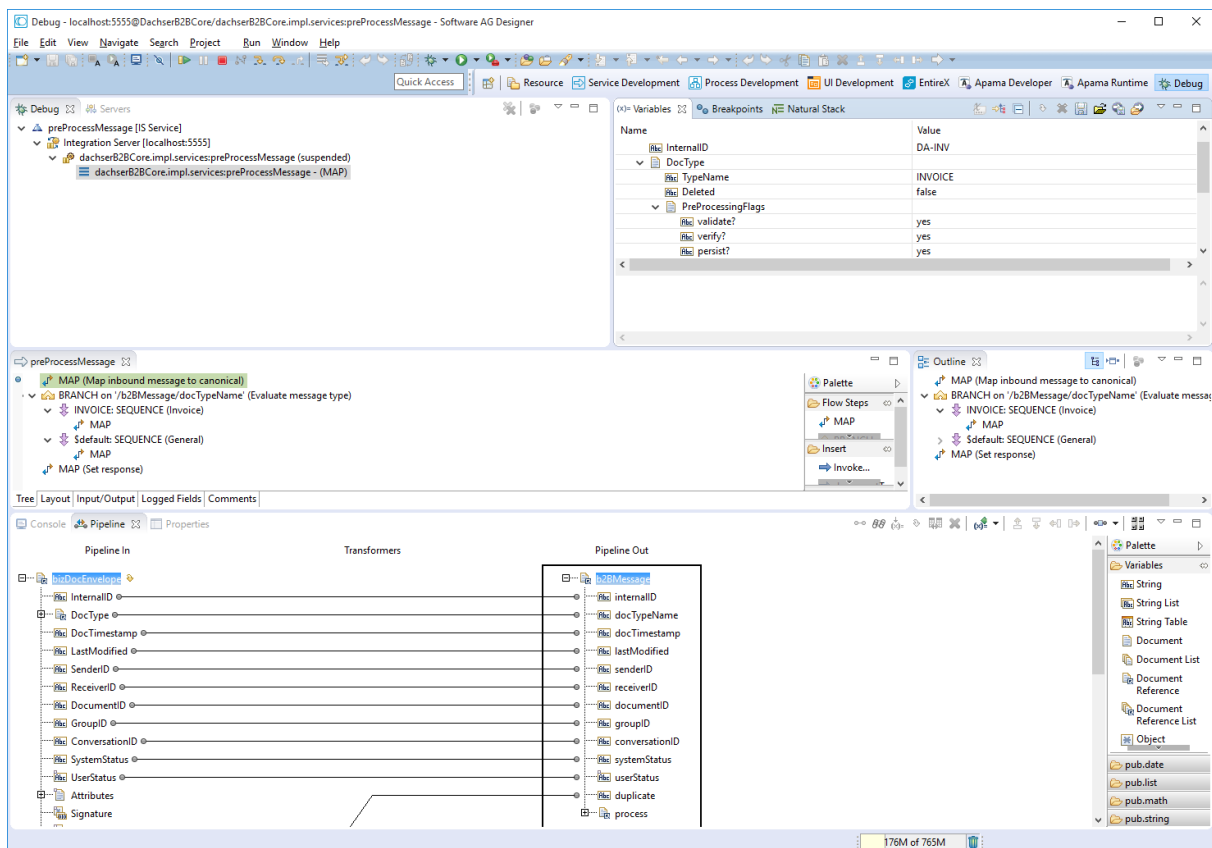


Figure 12: Debugging of Flow services in Software AG Designer's Debug perspective

When developing Integration Server assets such as Flow services or data structures, you can also compare two assets to see differences in their implementation, configuration and documentation. The according compare editor shows the assets side-by-side and highlights differences with according icons and hints. You can then decide which of the individual changes you would like to merge into the other asset. This can be done bi-directionally. The following screenshot shows the compare editor highlighting changes between two Flow services.

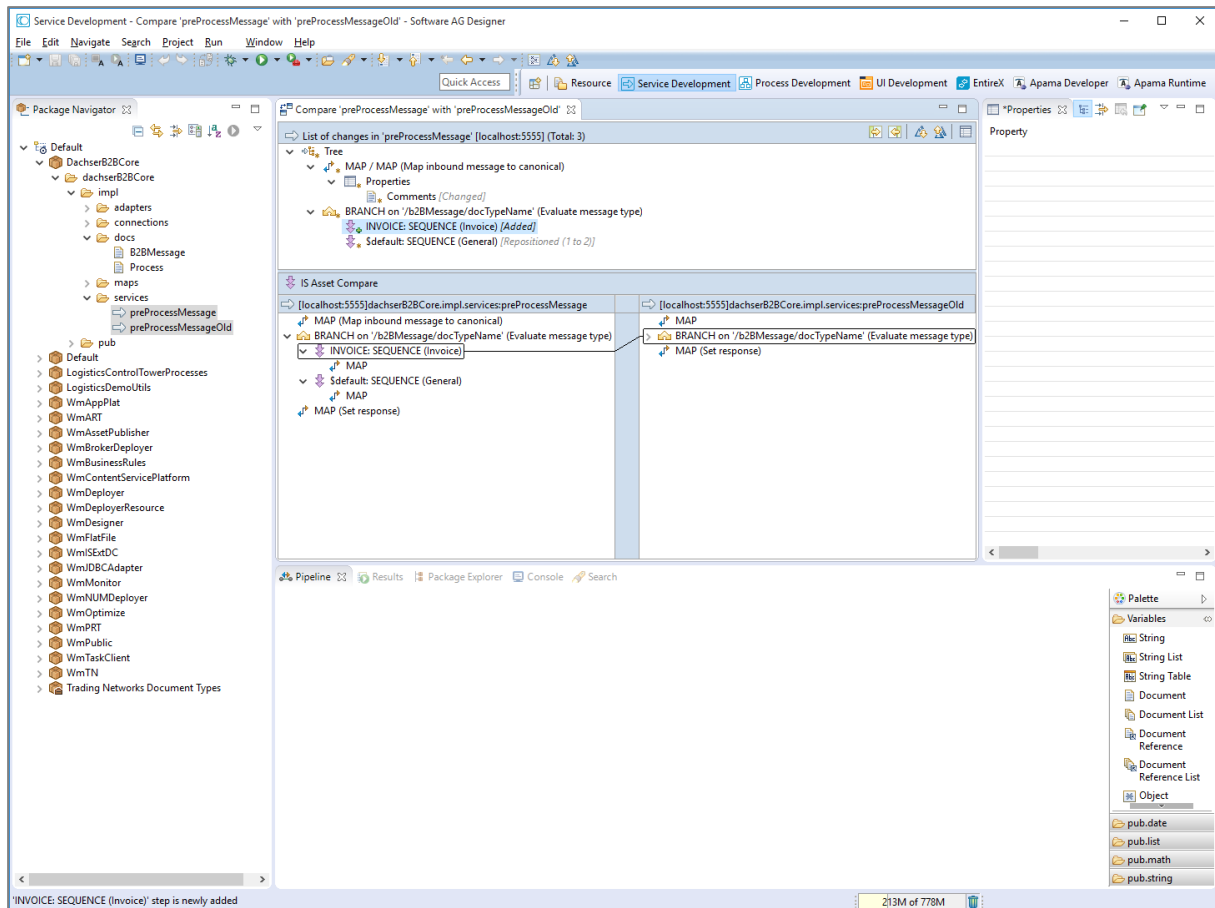


Figure 13: Compare and merge view for Integration Server assets such as Flow services

4.1.2 Service Enablement – Adapter

As mentioned earlier, Integration Server hosts also the Adapter Runtime. As such, Integration Server can leverage an extensive library of adapters allowing easy integration with different enterprise applications and technologies even through proprietary interfaces. Adapters are plug-ins for Integration Server enabling you to simply configure adapter services and adapter notifications. Adapter services are invoked on-demand whereas adapter notifications are listeners allowing for an event-driven integration with the backend application. This means no coding for back-end integration and significantly reduced development effort.

All adapters follow a unified pattern. They implement the often proprietary interface of the underlying application and hide it from the service developer. Hence, integration developers don't need to know the specifics of the application specific interface. As a consequence, all adapters share a standardized user interface, intelligent introspection¹ of data sources as well as common

¹ Intelligent introspection means that the adapter analyses the backend application and displays the interface details in the graphical configuration wizard of the adapter. In case of the JDBC adapter for database integrations, the wizard lists all tables that the connection's user is allowed to see. Once a table has been selected, the available columns and their data types are listed. The developer can then choose which columns are to be used as input variables and which columns are to be used as output variables for the adapter service. In case of an SAP R/3 backend, the according SAP adapter lists all RFC-enabled functions, assuming the integration is done via RFC instead of IDOC. The list of RFC-enabled functions can be filtered

connection pooling and session management capabilities. Thus, all adapters provide a common user experience for both administrative and development users.

As an example, the following figure shows the codeless configuration of a JDBC adapter service.

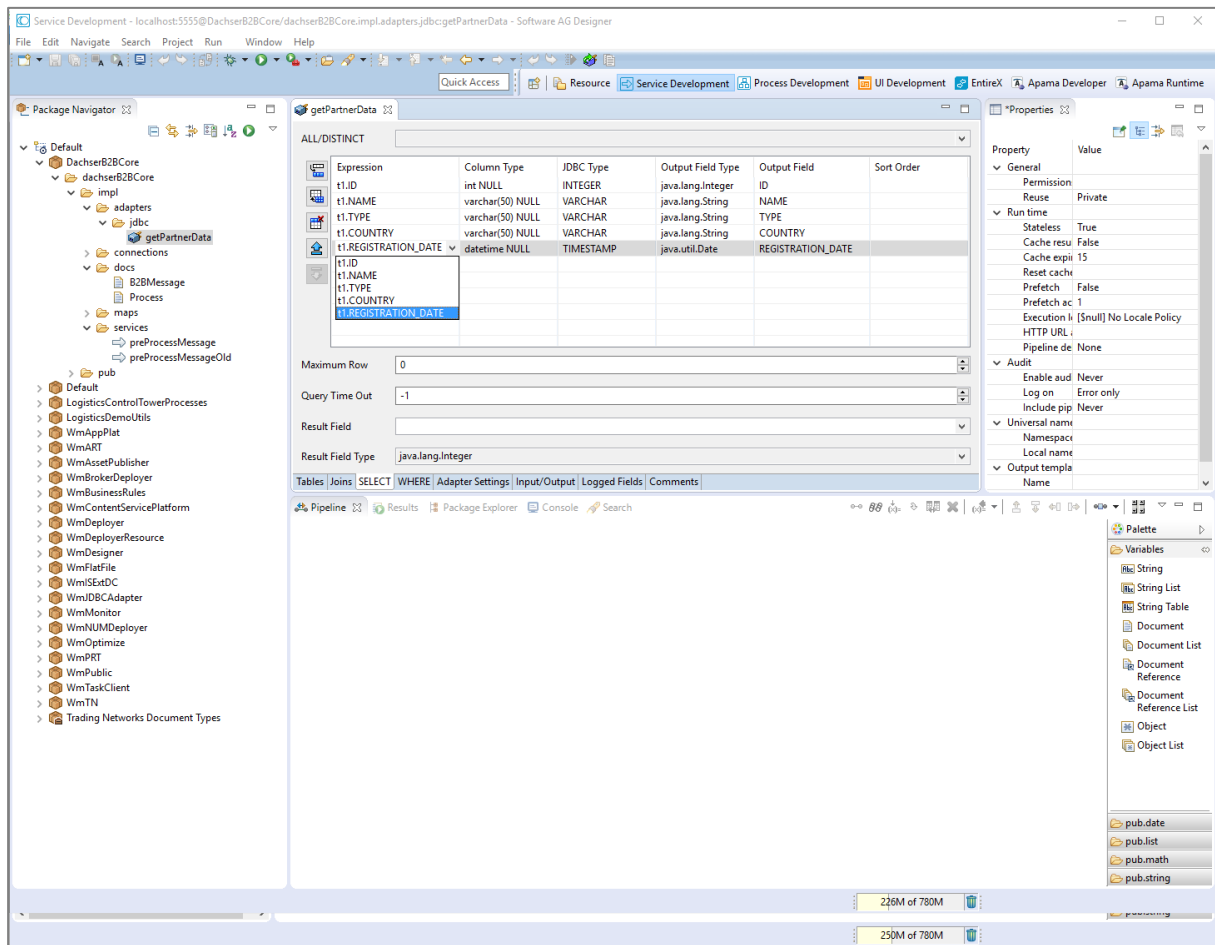


Figure 14: Creation of a JDBC Adapter Service using the codeless configuration facility

The adapters are licensed on an individual basis and are available for various packaged applications, technologies and big data frameworks.

The following table shows a list of currently available adapters.





























Application Adapters			
	Ariba Supplier on Ramp		Oracle Applications
	BroadVision One-To-One Enterprise		Oracle PeopleSoft
	JD Edwards World Applications		Remedy
	Lotus Notes		SAP
	OnRamp for Commerce One		Siebel
Technology Adapters			
	AS/400		Oracle Tuxedo
	Enterprise Java Beans		SAP PI/XI
	EntireX		Sopera
	JDBC		Microsoft WebSphere MQ
	Microsoft .NET		X.400
	Microsoft MQ		
Big Data Adapters			
	Apache Hadoop		Apache Kafka
	Apache HBase		Cassandra
	Apache Hive		MongoDB
	Apache Impala		

Table 1: Integration Server Adapters – Easy integration with various applications and technologies

If for an application an adapter is not available and integration with the application via standard interfaces is either not possible or not desired, Integration Server comes with an Adapter Development Kit providing capabilities to custom build an adapter sharing the same infrastructure, behavior as well as look and feel as the already existing adapters.

4.1.3 Service Enablement – Web Services

When maximum availability and standards based integration are important, then web services are a key communication channel to be used.

Integration Server provides support for common web service styles like SOAP and REST. There is built-in support for various standards like Web Service Interoperability (WS-I), Web Service Security (WSS), WS-Addressing, WS-ReliableMessaging, MTOM/XOP, and others. Any service on the Integration Server can be enabled as a web service with only a few configuration steps. Again no coding is required, making integration based on web services simpler and faster than ever before.

You also have the flexibility to follow both a Contract (WSDL) First or Service First approach for creating web service providers.

With the Contract (WSDL) First approach you start off by importing an existing WSDL file which defines the endpoint, bindings, interfaces, operations, and data types. You then enrich the generated Web Service Descriptor (WSD) with the actual implementation of the operations.

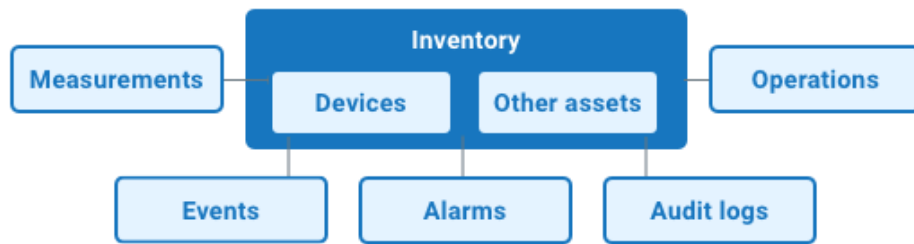
With the Service First approach you start off from any existing service (Flow, Java, etc.) on Integration Server to be used as the implementation of a web service operation, and generate the Web Service Descriptor (WSDL) based on that service. In both cases the entire approach is wizard-driven and codeless.

Consuming existing web services is simply achieved by importing the existing WSDL into Integration Server using Designer. All connectors and data models as defined in the WSDL are generated for you and the web service can be used straight away.

For REST resources you can also create REST API descriptors as a way of describing the operations provided by one or more REST resources together with information about how to access those operations, the MIME types the resources consume and produce, and the expected input and output for the operations. Fundamentally, a REST API descriptor is composed of REST resources and information about how to access those resources. Using this information, Integration Server creates and maintains a Swagger document for the REST API descriptor. Integration Server generates the Swagger document and can also import Swagger to follow the API-first development approach.

5 Cumulocity Domain Model

The following image shows the relevant aspects for devices and asset in an IoT application:



- 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. The model refers to them as *managed objects*. A managed object consists of a unique identifier, a type string, a time stamp showing the last update, a name and additional, custom fragments. This approach also enables developing generic application components. A specific kind of a managed object is a **device**. A device contains a special fragment "c8y_device". This is mostly used by the standard Cumulocity application to differentiate between devices and all other kinds of assets. Besides an internal identifier, so called "external IDs" help to identify managed objects. A managed object could have multiple external IDs. Typical examples of an external ID are serial number or MAC address.
- The inventory model supports two default hierarchies of objects: A communication hierarchy ("childDevices") and an asset hierarchy ("childAssets"). The two hierarchies 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 and tracks how devices are linked to the M2M platform from a communication point of view. Object hierarchies are not required to form a tree, the same asset can be a child of multiple parent assets.
- **Measurements** contain numerical data produced by sensors (like temperature readings) or calculated data based on information from devices (service availability of a device). Based in measurements a historical time series of numerical data could be produced.
- **Events** contain other (real-time) information from the sensor network, such as the triggering of a specific sensor.

Cumulocity has standardized representation of common devices and sensors as well as concepts for flexibly extending and modifying this representation. As a result, IoT 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.

5.1 Inventory

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

Managed objects can be in general “smart objects” such as smart electricity meters, home automation gateways and GPS devices. In our cases they will be hospitals.

The following JSON code shows an example of a managed object in the inventory.

```
{ "name": "Alice-Hospital Darmstadt",  
  "c8y_Position": { "lng": 8.6624854, "lat": 49.8774004 },  
  "c8y_IsDevice": {},  
  "IsKrankenhaus": {},  
  "type": "Krankenhaus",  
  "Bundesland": "Hessen" }
```

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 switch with the “managedObject” reference).

```
{ "name": "Alice-Hospital Darmstadt",  
  "c8y_Position": { "lng": 8.6624854, "lat": 49.8774004 },  
  "Bettenkapazitaet": {  
    "Normalbetten": {  
      "Soll": 100, "Max": 110, "Belegt": 70, "DavonBelegtmitCovid": 30, "Frei": 30},  
    "ITSBettenBMinus": {  
      "Soll": 100, "Max": 110, "Belegt": 70, "DavonBelegtmitCovid": 30, "Frei": 30},  
    "ITSBettenBPlus": {  
      "Soll": 100, "Max": 110, "Belegt": 70, "DavonBelegtmitCovid": 30, "Frei": 30},  
  },  
  "c8y_IsDevice": {},  
  "IsKrankenhaus": {},  
  "type": "Krankenhaus",  
  "Bundesland": "Hessen" }
```

In general, each managed object consists of

- A unique identifier that references the desired object.
- A type string that defines the object type.
- A time stamp showing the last update.
- Additional *fragments*.

5.1.1 Fragments

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:

```
{
  "name": "Alice-Hospital Darmstadt",
  "c8y_Position": {
    "lng": 8.6624854,
    "lat": 49.8774004
  },
  "Bed_Capacity": {
    "Normal": {
      "Target": 100,
      "Max": 110,
      "Occupied": 70,
      "OccupiedCovid": 30,
      "Free": 30
    }
  },
  "c8y_IsDevice": {},
  "IsHospital": {},
  "type": "Krankenhaus",
  "State": "Hessen"
}
```

In this example, a fragment “Bed_Capacity” identifies the different capacities of beds available or occupied within a hospital.

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.

This can on a later stage be used to add additional information such as availability of masks, staff or other material.

5.2 Events and Alarms

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

Events come in three types:

- A base event signals when something happens. An event can be triggered when for example a new dataset was uploaded.
- 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.

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 critical alarm that has been cleared:

```
{
  "severity": "CRITICAL",
  "creationTime": "2020-03-22T11:03:05.167Z",
  "count": 1,
  "history": {
    "auditRecords": [],
    "self": "https://t259528341.cumulocity.com/audit/auditRecords"
  },
  "source": {
    "name": "Alice-Hospital Darmstadt",
    "self": "https://t259528341.cumulocity.com/inventory/managedObjects/660",
    "id": "660"
  },
  "type": "ExperimentalIssue",
  "self": "https://t259528341.cumulocity.com/alarm/alarms/902",
  "time": "2020-03-22T11:03:05.083Z",
  "text": "Threshold reached",
  "id": "902",
  "status": "CLEARED"
}
```

Alarms can perfectly be used to indicate certain threshold exceeds together with Smart Rules or the Apama Analytics Builder.

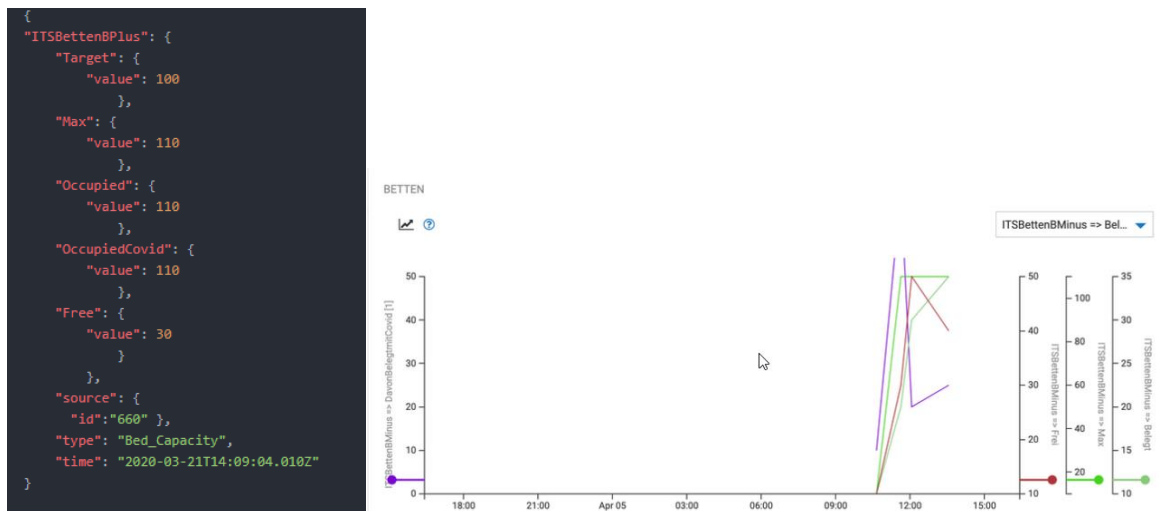
5.3 Measurements

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.

Measurements can be visualized in a time series basis such as e.g. a trend in time.

As an example of a measurement for intensive care beds and a point graph see attached screenshots.



6 Solution-Steps for Cumulocity (C8Y)

For rapid start the basic key steps of the solution on Cumulocity side are shown. Therefore a tenant creation as well as basic configuration of widgets are shown.

6.1 Tenant Creation

A tenant on SoftwareAG Cloud can be created easily via the signup page:

<https://signup.softwareag.cloud/#/?product=cumulocity>

Cumulocity IoT

Already have an account ? [Log in here.](#)

First name	Last name
<input type="text"/>	<input type="text"/>
Required	
Company	Phone number (optional)
<input type="text"/>	<input type="text"/>
Work email	
<input type="text"/>	
Country	
<input type="text" value="Germany"/>	
Cloud region	
<input type="text" value="Select region..."/>	
Required	
Environment name	<input type="text" value=".softwareag.cloud"/>
Username	
<input type="text"/>	
Password	
<input type="text"/>	
<input type="checkbox"/> I agree to the terms and conditions.	
<input type="checkbox"/> I acknowledge and accept, and have downloaded and validly countersigned, the Data Processing Agreement.	
<input type="checkbox"/> I consent that SAG Deutschland GmbH (Uhlandstraße 12, Darmstadt, 64297, Germany) contacts me via email and phone with information about the products, services and events offered by Software AG. I can withdraw this consent at any time – this can be done by unsubscribing here. Further information about how SAG Deutschland GmbH uses personal data can be found in the Privacy Statement.	

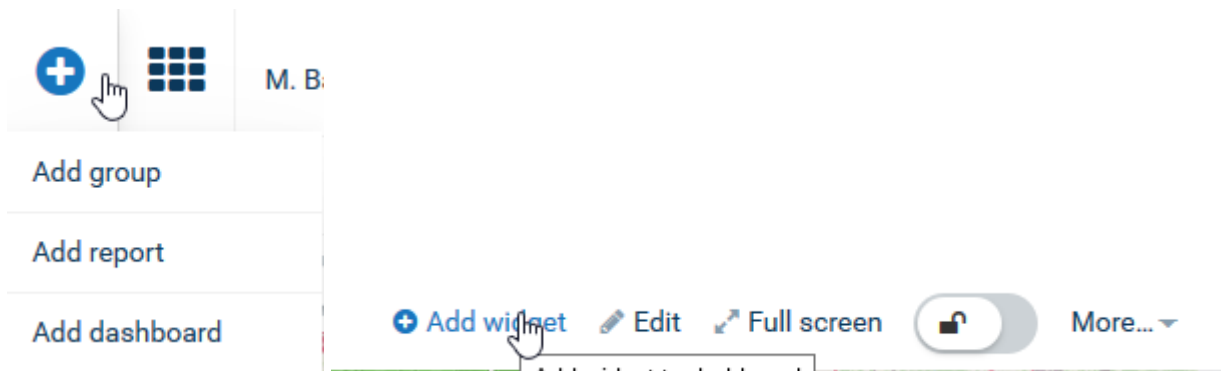
The tenant itself runs on AWS in Frankfurt in our public cloud and is limited to 30 days. The newly created tenant is available within minutes and brings all standard applications such as device management, cockpit and administration.

Since Cumulocity itself is platform independent the solution itself can also run in the edge or another public cloud.

6.2 Cockpit Adaptation

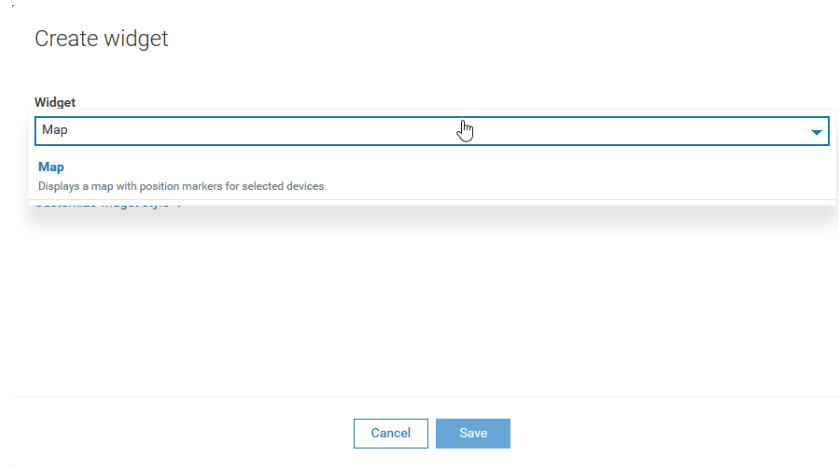
The Covidash Solution first of all visualizes just basic information such as lists of hospitals and their capacities. However, if provided, the localization on a map can be shown as well.

For the main dashboard the standard widgets were used. They can be placed within a group on a dashboard.

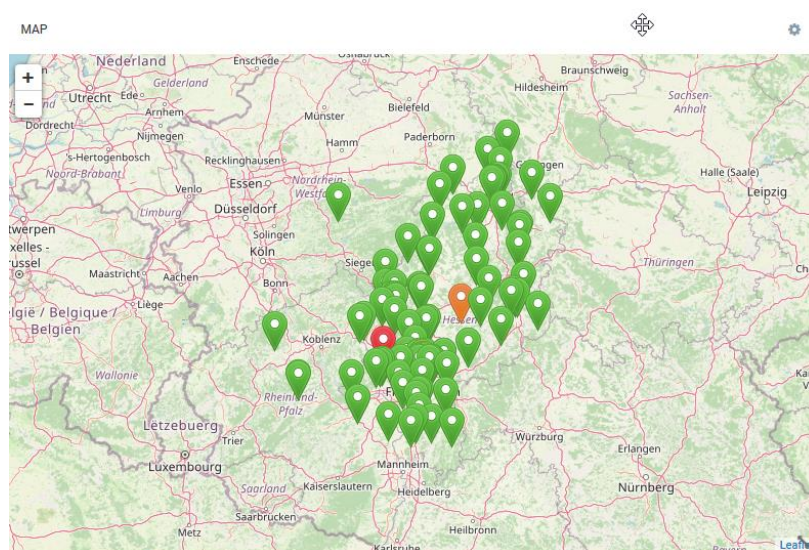


6.2.1 Map

The Map widget visualizes the devices or hospitals if the fragment “c8y_Position” has values.



It auto scales depending on the spread of the positions. Due to performance reasons the limit of devices on a map is set to 100 for the standard widget. This can however be changed due to an own created map widget.



The hospitals are also indicated as green, orange, yellow or red. This is depending on a certain alarm status of the hospital itself. If an alarm is active the colour of the position inside the map changes automatically.

6.2.2 Asset Table

The asset table visualizes the devices or hospitals and its fragments. It can be used to get a table overview of the different information inside the devices.

Create widget

Widget

Asset Table

Asset table
List of child devices of an asset with configurable columns.

Cancel

Save

Within the configuration the properties that want to be explicitly shown need to be added.

The fragment therefore needs to exist such that data needs to be pushed into Cumulocity before configuring the dashboard.

Create widget

Widget

Asset table

Title

Asset table

Target assets or devices

Krankenhäuser
Augsburg

Properties

+ Add property + Add action

Show	Label	Property	Render type
You must select at least one property.			

Customize widget style

Cancel

Save

The properties itself can be searched and later on activated or deactivated as well as sorted.

Select property

Show	Label	Property
<input type="checkbox"/>	Bundesland	Bundesland
<input type="checkbox"/>	Creation time	creationTime
<input type="checkbox"/>	ID	id
<input type="checkbox"/>	Last updated	lastUpdated
<input type="checkbox"/>	Name	name
<input type="checkbox"/>	Notes	c8y_Notes
<input type="checkbox"/>	Owner	owner
<input type="checkbox"/>	Type	type
<input type="checkbox"/>	Active alarms status	c8y_ActiveAlarmsStatus
<input type="checkbox"/>	Critical	c8y_ActiveAlarmsStatus.critical
<input type="checkbox"/>	Major	c8y_ActiveAlarmsStatus.major
<input type="checkbox"/>	Minor	c8y_ActiveAlarmsStatus.minor
<input type="checkbox"/>	Warning	c8y_ActiveAlarmsStatus.warning
<input type="checkbox"/>	Address	c8y_Address
<input type="checkbox"/>	City code	c8y_Address.cityCode
<input type="checkbox"/>	City	c8y_Address.city
<input type="checkbox"/>	Country	c8y_Address.country
<input type="checkbox"/>	Region	c8y_Address.region

Cancel Select

Edit widget

Properties

+ Add property + Add action

Show	Label	Property	Render type
<input checked="" type="checkbox"/>	Bundesland	Bundesland	Default
<input checked="" type="checkbox"/>	Name	name	Default
<input checked="" type="checkbox"/>	Last updated	lastUpdated	Default
<input type="checkbox"/>	Normalbetten	Bettenkapazitaet.Normalbetten	Default
<input checked="" type="checkbox"/>	Belegt	Bettenkapazitaet.Normalbetten.Belegt	Default
<input checked="" type="checkbox"/>	Frei	Bettenkapazitaet.Normalbetten.Frei	Default
<input checked="" type="checkbox"/>	Max	Bettenkapazitaet.Normalbetten.Max	Default
<input checked="" type="checkbox"/>	Soll	Bettenkapazitaet.Normalbetten.Soll	Default
<input checked="" type="checkbox"/>	DavonBelegtmitCovid	Bettenkapazitaet.Normalbetten.DavonBelegtmitCovid	Default

Customize widget style

Cancel Save

The placed asset table thus shows all the activated properties. Depending on the amount of devices or hospitals the loading speed of the webpage can decrease.

ASSET TABLE

BUNDESLAND	NAME	LAST UPDATED	BELEGT	FREI	MAX	SOLL	DAVONBELEGTMITCOVID
Hessen	Kinderklinik Darmstadt - Haupthaus	21/03/2020, 20:10:19	70	30	110	100	30
Hessen	Kerckhoff Klinik	21/03/2020, 20:10:27	70	30	110	100	30
Hessen	Kreiskrankenhaus Schotten	21/03/2020, 20:10:27	70	30	110	100	30
Hessen	Mathilden-Hospital	21/03/2020, 20:10:28	70	30	110	100	30
Hessen	Dalberg Klinik AG	21/03/2020, 20:10:28	70	30	110	100	30
Hessen	Klinikum F-Hoechst	21/03/2020, 20:57:05	70	30	110	100	30
Hessen	Markus Krankenhaus	21/03/2020, 20:57:07	70	30	110	100	30
Hessen	Universitaetsklinikum Frankfurt	21/03/2020, 21:01:31	70	30	110	100	30
Hessen	Asklepios Klinik	21/03/2020, 21:01:34	70	30	110	100	30

6.2.3 Alarm List

The alarm list visualizes all alarms on devices or hospitals. Since alarms can have different severities this can be used to have certain escalation schemes. Alarming can be controlled via Smart Rules very easily.

Create widget

Widget

Alarm

Alarm list

Displays a list of alarms filtered by object, severity and status

All critical alarms

Displays active critical alarms from all devices

Recent alarms

Log of recent alarms from all devices with any severity and status

Cancel

Save

Within the configuration it can be configured what severities or status should be shown such that also cleared alarms from the past can be visualized. Since alarms have the concept of types as well this can be used to show only alarms of a certain type such as e.g. “No Masks” or “No Beds”.

Create widget

Target assets or devices

Krankenhäuser

Augsburg

Status

☐ ACTIVE
☐ ACKNOWLEDGED
☐ CLEARED

Types

Alarm type

+ Add alarm type

Severities

☒ WARNING
☐ MINOR
☐ MAJOR
☐ CRITICAL

Order

☒ By active status
☐ By severity

Customize widget style

Cancel

Save

ALARM LIST

Threshold reached

22 March 2020 13:26

Marlen Krankenhaus

STATUS

ACTIVE: triggered 25 days ago

Type ExperimentalIssue

CHANGE LOG

Server time

22 March 2020 13:26

Alarm created

murat.bayram@softwareag.com

Threshold reached

22 March 2020 13:22

HELIOS-Klinik Idstein

Threshold reached

22 March 2020 13:26

Kreiskrankenhaus Weilburg

Threshold reached

22 March 2020 12:03

Alice-Hospital Darmstadt

6.3 Smart Rules

Cumulocity IoT comes with a built-in rule engine to analyze data in real-time and to perform actions based on these insights. To easily create rules the Cockpit application includes a Smart Rules builder which allows you to create rules from templates (so-called smart rule templates). Those Smart Rules are running either in a global context (one rule for all device types) or in a local context (one rule just for this specific device or group).

The component that realizes the abovementioned smart rules is our high-performance streaming analytics engine Apama. Apama is a market-leading platform for streaming analytics and intelligent automated action on fast-moving big data. Combining event processing, messaging, in-memory data management and visualization, this platform is the most complete solution to turn relentless data streams—like those produced by the Internet of Things (IoT)—into meaningful real-time metrics.

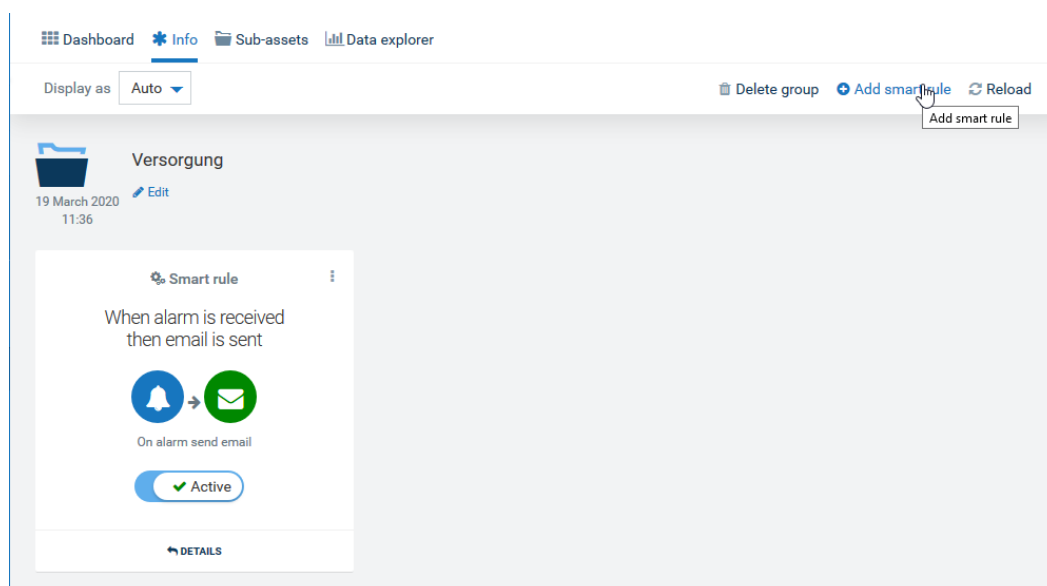
Using the streaming analytics engine, various root cause analysis methods can be implemented. Alarms managed by Cumulocity IoT platform can be extended with root cause information, or non-root cause alarms can be cleared. Typical root cause methods include:

- Time window based: Aggregate different faults based on time
- Filter short lived, false alarms
- Use historical data to rate alarm severity

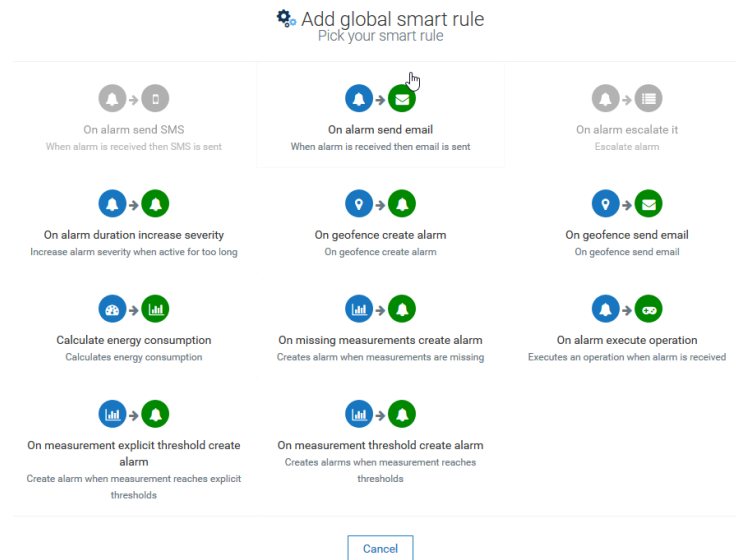
The Cumulocity IoT platform's streaming analytics engine can take multiple streams of data and look for patterns across the individual streams. New streams can be created and/or actions taken when patterns are identified. The streaming data can also be enriched with historical context data that is not part of the live stream.

For a great user-experience handling, Apama Analytics Builder gives engineers and other domain experts a web-based drag-and-drop capability to develop analytic applications on streaming data. They can test their applications with simulated or live data and deploy them immediately with a single click for real-time processing of machine data and events.

Smart Rules are easy to use rules to create basic functionalities such as alarming on threshold etc. They can be placed within a group in the info tab or on a global level as well.



The Smart Rules are predefined workflows and just allow certain configurations.



6.4 Apama Analytics Builder

Using Apama streaming analytics, you can add your own logic to your IoT solution for immediate processing of incoming data from devices or other data sources. These user-defined operations can, for example, alert applications of new incoming data, create new operations based on the received data (such as sending an alarm when a threshold for a sensor is exceeded), or trigger operations on devices.

APAMA ANALYTICS BUILDER

– Key capabilities

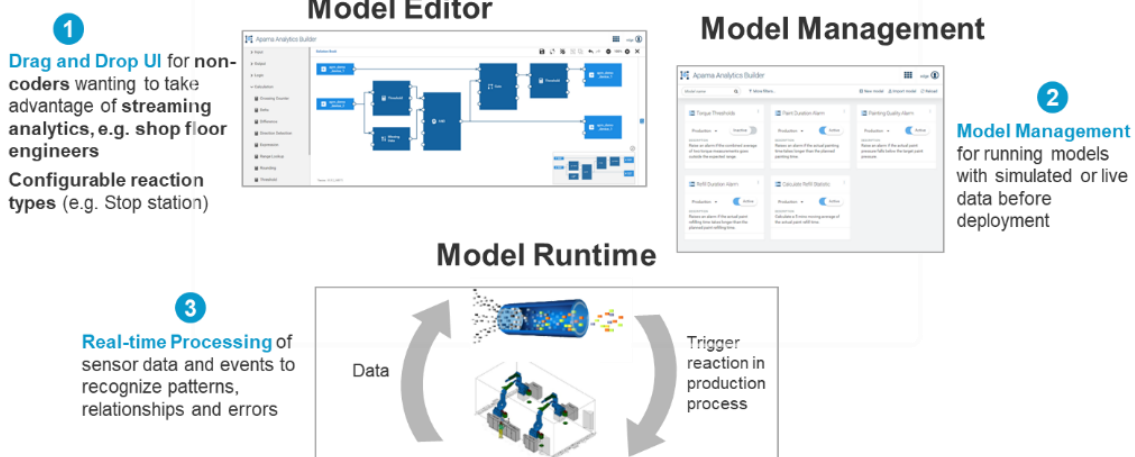


Figure 15: Apama supports laboratory staff in creating useful alerts by combining several sensor readings. More important, the easy to use UI brings transparency into complex rules and avoids competency bottlenecks.

The operation logic is based on Apama's Event Processing Language (Apama EPL).

Typical real-time analytics use cases include:

- Remote control: Turn a device off if its temperature rises over 40 degrees.
- Validation: Discard negative meter readings or meter readings that are lower than the previous.
- Derived data: Calculate the volume of sales transactions per vending machine per day.
- Aggregation: Sum up the sales of vending machines for a customer per day.
- Notifications: Send me an email if there is a power outage in one of my machines.
- Compression: Store location updates of all cars only once every five minutes (but still send real-time data for the car that I am looking at to the user interface).

There are multiple ways to integrate and develop streaming analytics depending on the background of the user.

- Apama EPL Apps -> User with coding background
- Apama Analytics Builder -> Coding blocks for people with no coding background

Apama EPL Apps is a web application which is available from the application switcher. It allows you to develop Apama applications directly within Cumulocity IoT, written in Apama EPL. You can also use it to import existing Apama applications (*.mon files) into Cumulocity IoT. When you then activate an Apama application, you deploy it to Cumulocity IoT.

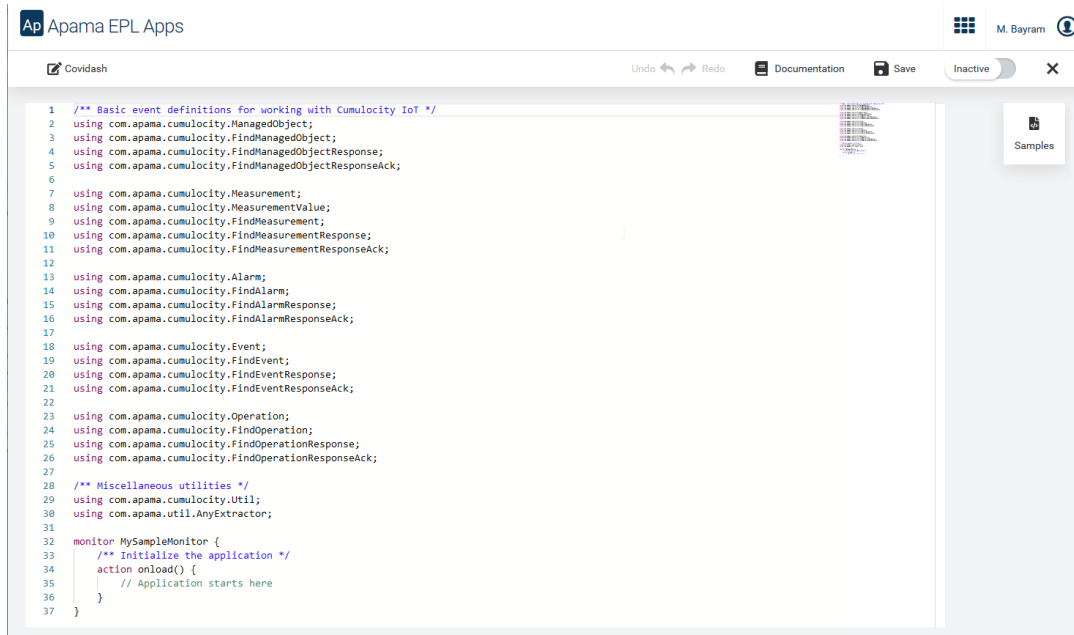


Figure 16: Apama EPL Apps and deployed code in EPL.

Apama Analytics Builder is a web application which is available from the application switcher. It allows you to build analytic models that transform or analyze streaming data in order to generate new data or output events. The models are capable of processing data in real time.

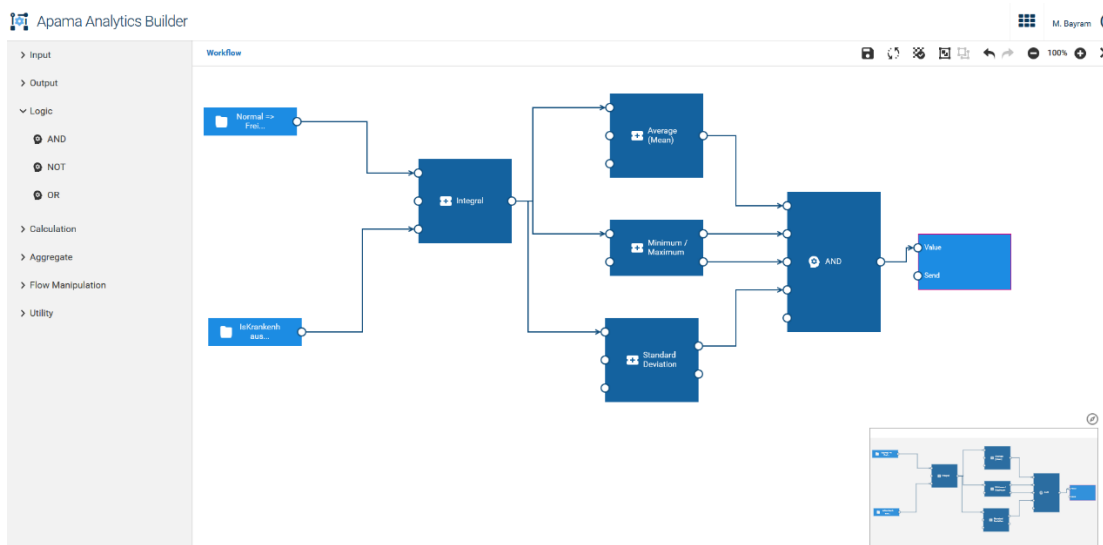


Figure 17: Graphical environment with blocks to create rules by logic and connection strings.

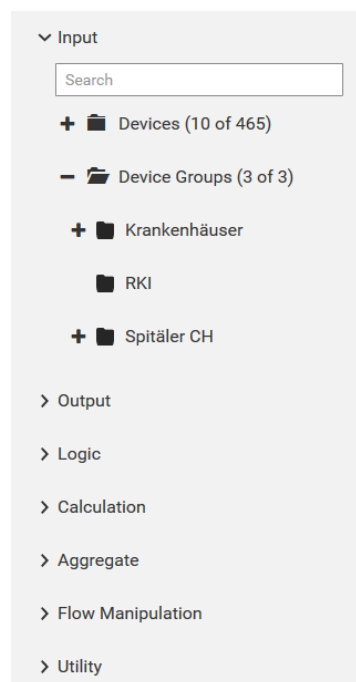
You build the models in a graphical environment by combining pre-built *blocks* into *models*. The blocks in a model package up small bits of logic, and have several inputs, outputs and parameters. Each block implements a specific piece of functionality, such as receiving data from a sensor, performing a calculation, detecting a condition, or generating an output signal. You define the configuration of the blocks and connect the blocks using *wires*. You can edit the models, simulate deployment with historic data, or run them against live systems.

The Apama Analytics Builder requires APAMA which might not be available in the trial version directly.



The Analytics Builder itself addresses self-service real time analytics and is thus easy to use without deep coding knowledge.

In order to create flows there is the concept of inputs, outputs and logic.



The different blocks can be used and combined.

The input therefore is configured in that way that inputs such as measurements, alarms, events or just managed object updates triggers the particular workflow on certain devices or groups.

Krankenhäuser

Input block for measurements or other inputs from a Cumulocity device or device group.

Parameters

Block Type
Measurement Input

Device or Device Group
Krankenhäuser

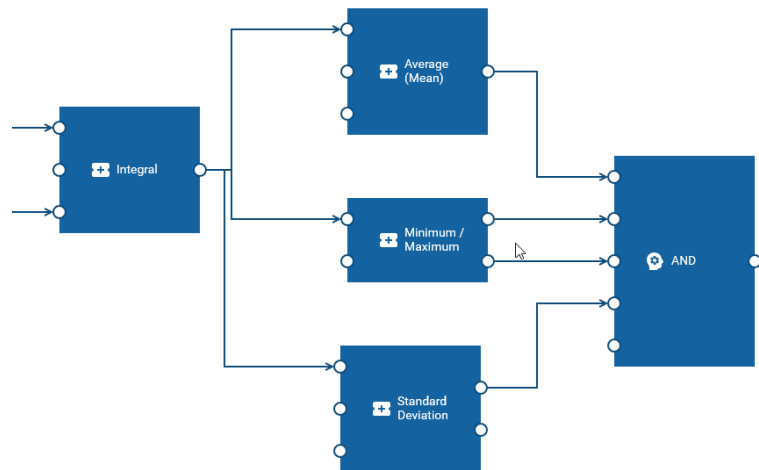
Cumulocity Fragment and Series
Select

☐ Ignore Timestamp

Duplicate

Delete

The logical block can be used to transform the event stream e.g. create an average or gives a standard deviation.



The output at the end defines what should be triggered at the end e.g. an alarm or an measurement.

Adullam-Stiftung Basel Adullam-Geriatriespital

Output block for creating measurements or other outputs for a Cumulocity device (or the triggering device).

Parameters

Block Type
Measurement Output

Device or Trigger Device
Adullam-Stiftung Basel Adullam-Geriatriespital

Fragment Name

Series Name

Unit - Optional

Duplicate

Delete

6.5 Bed Capacity Model

Modelling the bed capacity within Cumulocity allows not just the concept of fragments but also the later possibility to add additional hospital information such as availability of masks, staff and others. However the modelling was significantly influenced by the solution provider in Germany which are as an example illustrated in this section as well.

Beds in our approach follow 3 major categorizations:

1. Normal bed: This is a bed that does not require any intensive care. Of course there is staff and treatment involved but from a COVID-19 point of view these capacities are not highly important.
2. Bed ICU without Ventilation: These are beds where no direct ventilation is required but e.g. additional oxygen via masks and more intensive care such as advanced monitoring. From a COVID-19 point of view these are capacities where the status of a patient dramatically changes.
3. Bed ICU with Ventilation: These are beds where direct ventilation is required. These are highly cared patients and in terms of COVID-19 limited and important capacities.

A modelled hospital with these major categories is shown here.

```
{
  "name": "Alice-Hospital Darmstadt",
  "c8y_Position": {
    "lng": 8.6624854,
    "lat": 49.8774004
  },
  "Bed_Capacity": {
    "Normal": {},
    "ICU+": {},
    "ICU-": {}
  },
  "c8y_IsDevice": {},
  "IsHospital": {},
  "type": "Krankenhaus",
  "State": "Hessen"
}
```

Within these major categories are additional sub properties. Since not only free beds want to be shown extensions regarding e.g. occupancies are made in the following way:

1. Free: Available beds that are not occupied.
2. Target: This is the number of beds a hospital of the category has without a drop in medical care. Meaning this would be a fully occupied hospital that still operates well. It is not the maximum overall capacity.
3. Max: This is the maximum overall capacity. It extends the target capacity with an overcapacity such as beds in the floor or within waiting rooms. Quality of medical care changes.
4. Occupied: Occupied beds within the category.
5. Occupied with Covid: This are the occupied beds from patients that were positive tested on COVID-19.

```
{
  "name": "Alice-Hospital Darmstadt",
  "c8y_Position": { "lng": 8.6624854, "lat": 49.8774004 },
  "Bed_Capacity": {
    "Normal": {
      "Target": 100, "Max": 110, "Occupied": 70, "OccupiedCovid": 30, "Free": 30 },
    },
  "c8y_IsDevice": {},
  "IsHospital": {},
  "type": "Krankenhaus",
  "State": "Hessen"
}
```

As you will see in the IVENA and Rescuetrack examples there are many variations within the different federal states in Germany. The very generic model used here does e.g. not distinguish between adults and kids since the data is not available overall. However if the data does exist this might be something that should be modelled as well.

Measurements are used in order to show a time series of the development of capacities. For each of the major categories a measurement needs to be send.

```
{
  "Normal": {
    "Total": {
      "value": 100
    },
    "Max": {
      "value": 110
    },
    "Belegt": {
      "value": 110
    },
    "DavonBelegtmitCovid": {
      "value": 110
    },
    "Frei": {
      "value": 30
    },
    "source": {
      "id": "660"
    },
    "type": "Bettenkapazitaet",
    "time": "2020-03-21T14:09:04.010Z"
  },
  "ITSBettenBPlus": {
    "Target": {
      "value": 100
    },
    "Max": {
      "value": 110
    },
    "Occupied": {
      "value": 110
    },
    "OccupiedCovid": {
      "value": 110
    },
    "Free": {
      "value": 30
    },
    "source": {
      "id": "660"
    },
    "type": "Bed_Capacity",
    "time": "2020-03-21T14:09:04.010Z"
  }
}
```

In this case a normal and a ICU with ventilation bed measurement are send. The capacity is packed into the field value within the fragment of the sub-categories. That way the standard widgets can be used since they are watching for the value field.

6.5.1 IVENA - Example

IVENA is the largest solution provider in Germany and covers 8 federal states. The hospitals are sub-grouped into the responsible central office that does the planning and alarming of e.g. ambulance cars.

It is structured via tables and uses a SOAP Interface. This is an example of the bed capacity object within their solution.

ICU-Betten mit Beatmung - KHS gesamt		IMC-Betten ohne Beatmung - KHS gesamt		Normalpflegebetten ohne ICU und IMC - KHS gesamt		Anzeige aktualisieren		
Sonderlage: ICU-Betten mit Beatmung - KHS gesamt								
Integrierte Leitstelle Nürnberg	Soll-ICU-Betten	Max.-ICU-Betten (Soll-Betten mit Not-Beatmung)	Belegte ICU-Betten	davon belegte ICU-Betten mit COVID	davon belegte ICU-Betten mit ECMO	Anz. freie ICU-Betten	davon freie ICU-Betten für COVID	davon freie ICU-Betten mit ECMO

On the top are the 3 different tables that are used for the major categories. Within this tables are the columns of max, free, occupied etc.

6.5.2 Rescuetrack – Example

The Solution provider resuretrack uses a REST-API. Within Germany 3 federal states are connected to this solution.

This is an example of the bed capacity object within their solution.

<pre>{ "id": "00000000-0000-0000-0000-000000000000", "name": "Intensiv o. Beatmung", "shortName": "Intsv. o. Btmg.", "capacity": { "free": 5, "total": 36 }, "state": "yellow", "stateCount": { "red": 30, "green": 6 }, "info": "", "accessMode": "read", "lastChanged": "2020-03-21T08:09:58.8428769Z" }</pre>	<pre>{ "id": "00000000-0000-0000-0000-000000000000", "name": "Hosp. COVID Pat.", "shortName": "COVID-Pat", "capacity": { "free": 6, "total": 6 }, "state": "green", "stateCount": { "green": 6 }, "info": "", "accessMode": "read", "lastChanged": "2020-03-20T05:53:49.365Z" }</pre>
--	---

On the left here is a ICU without ventilation and a normal bed occupied with COVID-19. Rescuetrack itself categorizes COVID beds as a major type and only clusters its properties into free and total.

7 Interface to Cumulocity

7.1 REST

This section describes specific aspects using the REST interface.

Cumulocity employs REST for all external communication. Regardless whether the communication originates from IoT devices, from web applications or from back-office IT systems, the communication protocol is always REST.

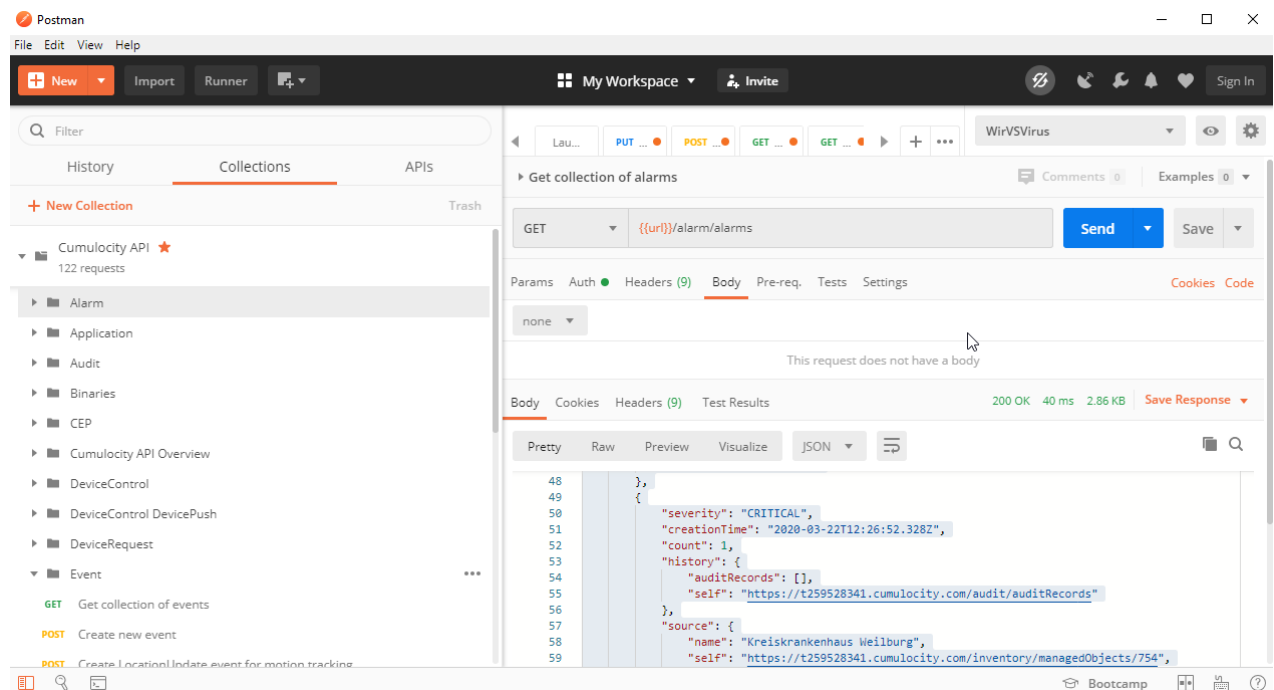
REST is a very simple and secure protocol based on HTTP(S) and TCP. It is today the de-facto Internet standard supported by all networked programming environments ranging from very simple devices up to large-scale IT. One of the many books introducing REST is RESTful Web Services.

With this API description you will learn how to use Cumulocity's REST interfaces to develop microservice applications on top of the Cumulocity platform.

7.2 Using the Interface and Postman Collection

Nowadays, most programming environments have particular support for REST-based communication. For experimentation and for understanding Cumulocity's REST interfaces, it is helpful to use one of the numerous available command line tools or browser extensions.

Graphical REST clients such as Postman are a convenient way to explore REST interfaces and the Cumulocity database content.



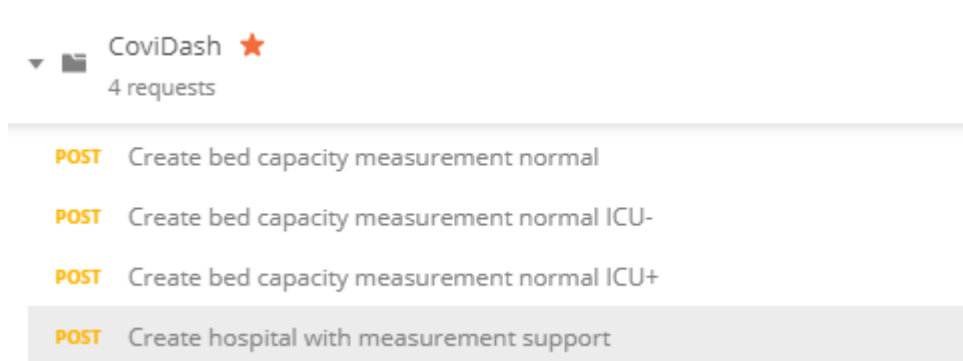
Cumulocity IoT provides numerous online API examples. You can import the APIs as a JSON from here: [Link](#)

Now, click the **Collections** tab on the top left of Postman. You should see a folder *Cumulocity API* with the examples. Open that folder and the sub-folder *Alarms*, then click **Get collection of alarms**. This shows an example on how to get alarms from Cumulocity IoT.

Note that the example contains placeholders, in this case a placeholder `{{url}}` in `{{url}}/alarm/alarms`. You need to tell Postman how to fill these placeholders and by this, how to connect to your Cumulocity IoT account. To do so, create an environment and configure the placeholders.

- Click on the cogwheel on the top right and choose **Manage Environments**, then click **Add**.
- Enter a name for the environment (e.g., your tenant ID), then add values for the placeholders.
- Configure a key `url` with a value of `https://<yourTenant>.cumulocity.com`. Click **Submit**.
- Configure a key `auth` with the value of the Authorization header for the REST requests.
- Click **Add**, then close the dialog. Now select your newly created environment from the drop-down box on the top right, that initially reads “No environment”.

For the purposes of CoviDash we provide a smaller API’s set. It consists of sending a Measurement and the creation of a hospital.



7.3 Device SDK

Cumulocity offers a wide range of functionality for interfacing IoT devices and other IoT-related data sources with the Cumulocity IoT platform.

This Device SDK guide provides detailed information on device integration using MQTT, REST and C++.

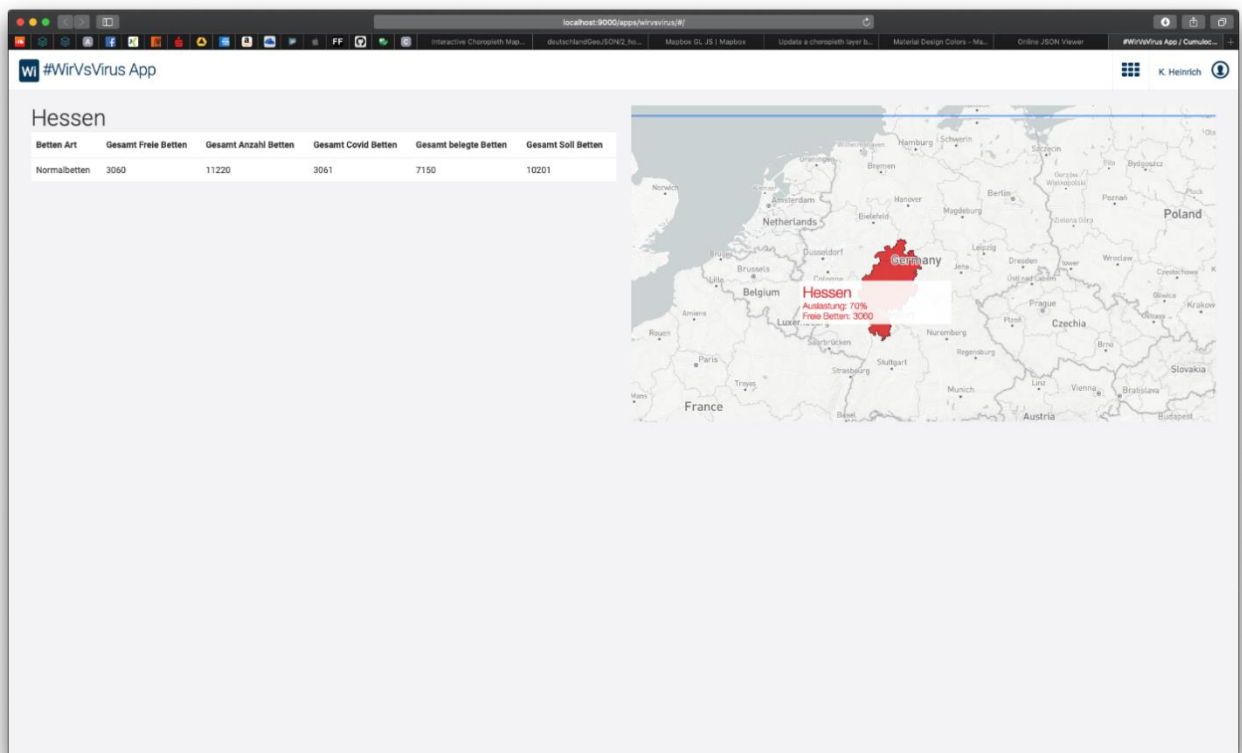
For interfacing devices with Cumulocity, we recommend using MQTT and SmartREST, which can be very efficiently implemented using available MQTT client libraries such as Eclipse Paho.

An up-to-date open source reference implementation of a Cumulocity agent for embedded Linux systems with many device management features can be found at [bitbucker](#).

7.4 Web SDK

Cumulocity provides an Angular-based WebSDK to allow modifying or extending the default UI applications (Cockpit, Device Management and Administration). Additionally, Cumulocity provides a NodeJS-based library to develop fully custom UI applications using an alternative technology stack (e.g. React, VUE.js, ...). The following subsection briefly examples the Angular-based WebSDK. Please note, more details information about the Web SDK can be found in the official documentation.

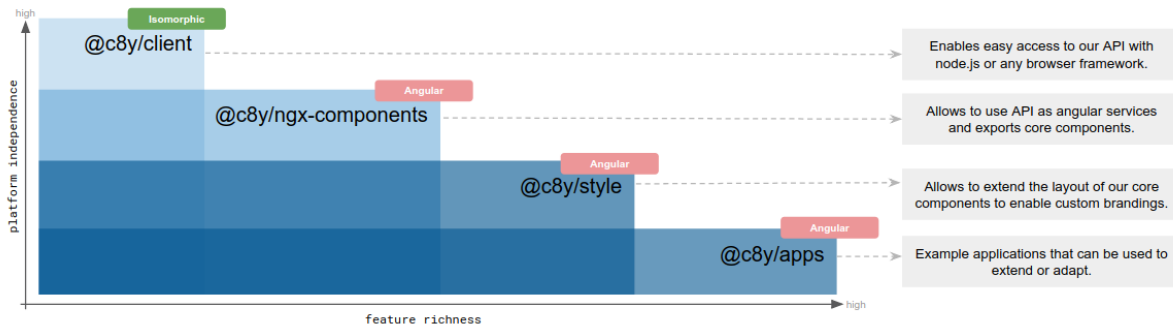
Within the Hackathon we were able to quickly generate a whole new application that is mainly based on a visualisation occupancy within states. The Web SDK allowed a rapid application generation. Since the code strongly bases on the german federal structure the layout and the queries might change. However web developers can easily generate new web applications and upload them via CLI tools or through the UI itself.



This Web SDK enables you

- to develop web applications that can be deployed to the platform,
- to communicate authenticated with our API,
- to apply default or branded UI components to your custom application.

It consists of the following packages deployed to npm in the scope @c8y and available under Apache 2.0 license:



These packages depend on each other from top to bottom. While the @c8y/client is a very low-level API interface with nearly no dependencies, the @c8y/apps provide feature rich applications by including @c8y/ngx-components and @c8y/client.

The goal of these splitting is to provide the right package for every use case, e.g. if you want to build a small application with React you could use the @c8y/client to do the API interaction. If you need a brandable feature rich application which is close to our Cockpit or Device Management application, you could use @c8y/ngx-components together with @c8y/styles.

Following is a list which explains the use cases of each package.

- @c8y/client: Use this client to access our API. The client is isomorphic, that means it could be used in node.js and in the browser.
- @c8y/ngx-components: A components collection and data access layer for Angular applications. This package can be used to build Angular applications.
- @c8y/styles: The styles for the look & feel of an application. Extend this package to apply a custom branding to your application.
- @c8y/apps: Example and bootstrapping applications to easily let you start with the Web SDK.

You can find all our packages on npm. To quickly get you bootstrapped with these packages we have built an CLI tool called @c8y/cli.

8 CoviDash Implementation in webMethods Integration Server (IS)

In the scope of the German Hackathon, webMethods Integration Server established the link between the German regional solution providers and Cumulocity.

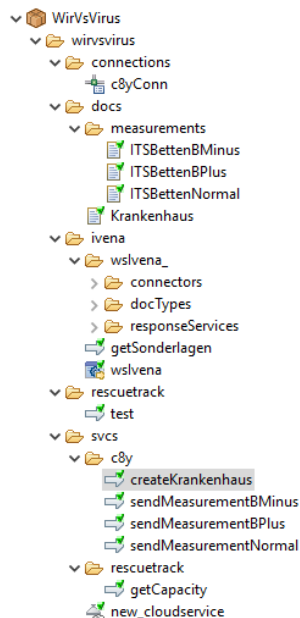


Figure 18: webMethods Integration Server Package structure

Integration Server consumes APIs from the regional solution providers, transforms this data based on doc types via mappings, and pushes the data to Cumulocity using Cumulocity's standard REST API.

Integration Server also ensures all necessary security aspects between German regional solution providers and Cumulocity.

As the German regional solution providers, like IVENA and rescuetrack, are specific to Germany, and their APIs are their protected by law intellectual properties, our created webMethods Integration Server connectors and services could not be exposed to public.

On request, sample connectors could be published, covered by an Apache License.

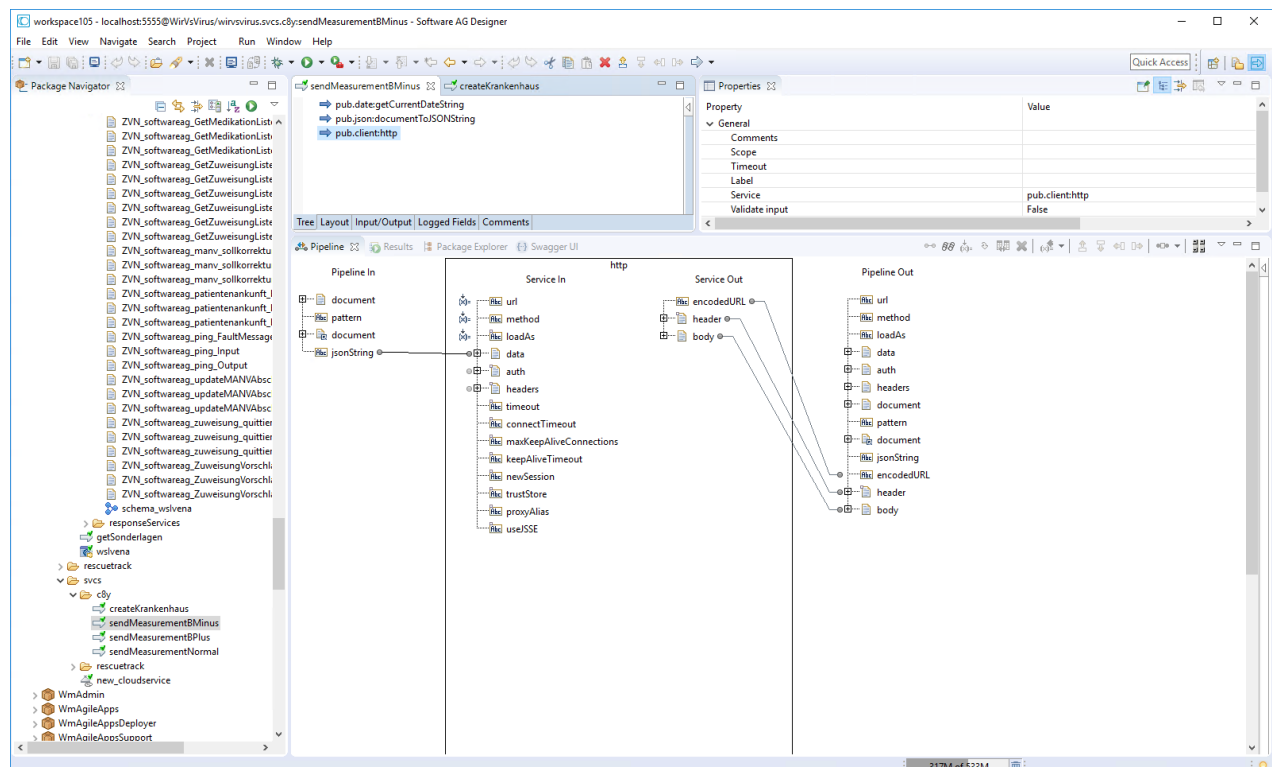


Figure 19: Software AG Designer: Example webMethods Integration Server data transformation mapping

SAG Deutschland GmbH

Uhlandstraße 9
64297 Darmstadt

T: +49 6151 92-3100
F: +49 6151 92-3666
E: info@softwareag.com

www.SoftwareAG.com

About Software AG

Software AG (Frankfurt TecDAX: SOW) helps companies with their digital transformation. With Software AG's Digital Business Platform, companies can better interact with their customers and bring them on new 'digital' journeys, promote unique value propositions, and create new business opportunities. In the Internet of Things (IoT) market, Software AG enables enterprises to integrate, connect and manage IoT components as well as analyze data and predict future events based on Artificial Intelligence (AI). The Digital Business Platform is built on decades of uncompromising software development, IT experience and technological leadership. Software AG has more than 4,700 employees, is active in 70 countries and had revenues of €865 million in 2018. Learn more at www.SoftwareAG.com.

© 2019 Software AG. All rights reserved. Software AG and all Software AG products are either trademarks or registered trademarks of Software AG. Other product and company names mentioned herein may be the trademarks of their respective owners.