IoTwins
BIG DATA OPTIMIZE INDUSTRY AND SERVICES

Grant Agreement N°857191

Distributed Digital Twins for industrial SMEs: a big-data platform

DELIVERABLE 2.2 – IMPLEMENTATION OF THE IOTWINS PLATFORM (I)

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement № 857191

Document Identification

Project	IoTwins
Project Full Title	Distributed Digital Twins for industrial SMEs: a big-data platform
Project Number	857191
Starting Date	September 1st, 2019
Duration	3 years
H2020 Programme	H2020-EU.2.1.1. - INDUSTRIAL LEADERSHIP - Leadership in enabling and industrial technologies - Information and Communication Technologies (ICT)
Topic	ICT-11-2018-2019 - HPC and Big Data enabled Large-scale Test-beds and Applications
Call for proposal	H2020-ICT-2018-3
Type of Action	IA-Innovation Action
Website	iotwins.eu
Work Package	WP2 - IoT-Edge-Cloud infrastructure and big data services for SMEs
WP Leader	FOKUS
Responsible Partner	FOKUS
Contributing Partner(s)	BSC, INFN, ESI, SAG, SAGOE, THALES, UNIBO
Author(s)	D. Nehls (FOKUS)
Contributor(s)	C. Garcia (BSC), C. Ahouanonou (ESI), D. Cesini (INFN), B. Martelli (INFN), T. Preclik (SAG), T. Schüle (SAG), S. Busch (SAG), F. Kintzler (SAGOE), V. Thouvenot (THALES), A. Stoian (THALES), K. Kapusta (THALES), J. G. Di Modica (UNIBO)
Reviewer(s)	C. Arlandini (CINECA), V. Gowtham (TUB)
File Name	D2.2 – Implementation of the IoTwins platform (I)
Contractual delivery date	M14 – 31 October 2020
Actual delivery date	M15 – 24 November 2020
Version	1.2
Status	Final
Type	R: Document, report
Dissemination level	PU: Public
Contact details of the coordinator	Francesco Millo, francesco.millo@bonfiglioli.com

Document log

Version	Date	Description of change
V0.1	15/07/2020	Initial Document structure
V0.2	01/09/2020	Revised structure, initial content collection
V0.3	28/09/2020	Implementation of new document template
V0.4	29/09/2020	Created separate section for existing Software and Tools
V0.5	12/10/2020	Internal Review
V0.6	28/10/2020	Restructuring Requirements
V0.7	02/11/2020	Consortium Review
V1.0	13/11/2020	Further version elaborated considering the comments received from project partners
V1.1	18/11/2020	Pre-final version sent to the consortium for a final check
V1.2	24/11/2020	Final version

Executive summary

This deliverable gives an overview of the current state of the IoTwins Platform. The project's objective is to "design and implement a platform that will provide a single point of access to heterogeneous computing, high capacity storage and interconnected resources." and in the same time enables "processing of big data coming from distributed sensors" orchestrated on the cloud, close to the data sources or distributed between both". This platform acts as a backbone infrastructure and testing framework for the whole IoTwins project: in particular, it offers itself as the basis for the execution of AI service components developed in the course of the project.

Given the requirements provided in Deliverable 2.1: "Architecture, and Technical and User Requirements", common software development mechanics were used to create first a high-level architecture and layers that the platform should support. As this deliverable is labeled as confidential and not available to the public reader, a short overview of its result is presented, distinguished between capabilities requested by the testbeds and non-functional security requirements. Based on the requirements use cases from the perspective of different stakeholders (IoTwins User and IoTwins Service Developer)

A detailed architecture consisting of functional blocks for each level of application (IoT, Edge, Cloud) was based on this, to distributedly manage digital representations of physical assets in both the manufacturing and facility management domains. This includes distributed storage, device-edge-cloud networking for big data transfer (batching and streaming), and the distributed processing of data at appropriate places either at the edge(s) of the network – close to the facilities – or in the central cloud, overall depending on the requirements, for example, low latency, bandwidth, computation power, and data privacy regulations.

In the following step existing tools and software were collected and mapped to the function blocks, building a first collection of tools for the IoTwins toolbox. This selection of tools will build the outline for a prototype platform implementation.

This deliverable is intended as a snapshot of a living architecture, that will be adapted to future developments where needed Updates will be published in D2.4 (M26 - October 2021) and D2.6 (M36 - August 2022).

Table of Contents

1 Introduction

IoTwins aims at enabling SMEs in the manufacturing and facility services sector to access edge-enabled and cloud-based big data analysis services in order to create digital twins of their assets. These digital twins can help in improving the production processes and optimization of the management of their facilities.

In this deliverable the design process for the platform that will provide a single point of access to heterogenous computing, high capacity storage and interconnected resources will be described.

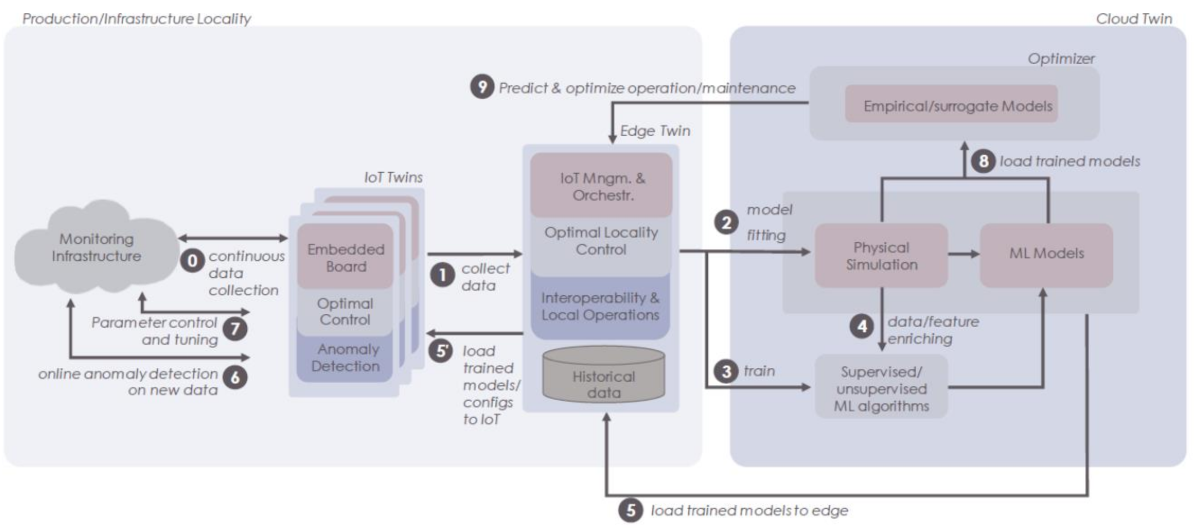
The platform will deliver interfaces to data analytics and AI techniques, physical simulation, optimizations and virtual lab services. In addition, the IoTwins platform will apply well established and emerging standards to enable communication- and data-level interoperability in the industrial IoT domain.

Therefore, in total 12 different example cases will be implemented during the course of the project at facilities of project partners (Table 1: Testbed Overview).

For the design of the architecture common software development methods have been applied.

Given the computing architecture schematized in Figure 1, requirements were collected from the use cases for the following layers of the IoTwins architecture:

- IoT layer (including the requirements for the interface between IoT and Edge layers)
- Edge Layer (including the requirements for the Interface between Edge and Cloud layers)
- Cloud Layer
- Application layer



Production/Infrastructure Locality
Cloud Twin
Optimizer
9 Predict & optimize operation/maintenance
Empirical/surrogate Models
Edge Twin
8 load trained models
IoT Twins
IoT Mngm. & Orchestr.
Optimal Locality Control
Interoperability & Local Operations
Historical data
Monitoring Infrastructure
0 continuous data collection
Embedded Board
Optimal Control
Anomaly Detection
1 collect data
2 model fitting
Physical Simulation
ML Models
Parameter control and tuning 7
online anomaly detection on new data 6
5' load trained models/ configs to IoT
4 data/feature enriching
3 train
Supervised/ unsupervised ML algorithms
5 load trained models to edge

Figure 1: IoTwins High Level Architecture

Based on the requirements identified, use cases are defined and presented in section 2.3.

Derived from these use cases a component-based overview architecture is created (section 2.4) which is then substantiated in more detail in section 3.

As presented in D2.1 the architecture is assumed to be layered based on the identified requirements. In this deliverable different function blocks have been generalized for each layer and existing solutions as well as known alternative approaches that fulfil equivalent function are described as the IoTwins Toolbox.

Software and tools already identified to be present in the toolbox are described in section 4.

After this summary of implementation, the deliverable concludes with a gap summary to identify missing parts in the existing setups and create the outline for future in-project developments.

Table 1: Testbed Overview

Testbed	Description
Manufacturing Testbeds	
TB1: Wind Turbine Predictive Maintenance	The testbed is aimed at creating a digital twin of a wind farm by aggregating simulation and Machine Learning (ML) models of single turbines for predictive maintenance.
TB2: Machine Tool Spindle Predictive Behavior	The testbed is aimed at creating multiple target-oriented digital twins of machine tools for the production of automotive components.
TB3: Predictive Maintenance for a Crankshaft Manufacturing System	In this testbed data is collected from a high throughput crankshaft manufacturing system to enable e.g. predictive maintenance for the production line.
TB4: Predictive Maintenance and Production Optimization for Closure Manufacturing	The objective of this testbed is to establish an overall management optimization including specific goals in the area of predictive maintenance.
Facility Management Sector	
TB5: Sport Facility Management and Maintenance	This pilot focuses on the management of facilities involving the flow of large crowds both during normal operation and during maintenance and upgrade phases involving construction.
TB6: Holistic Supercomputer Facility Management	The testbed focuses on data-driven prescriptive maintenance and optimization of large computing facilities.
TB7: Smart Grid Facility Management for Power Quality Monitoring	This testbed focuses on smart grid KPI computation as close to the data sources as possible, with input of higher-levels info, that cannot be accessed locally.
Replicability	
TB8: Patterns for Smart Manufacturing for SMEs	This use case is devoted to the definition of a general and replicable methodology for SMEs based on the convergence of data analytics, AI, IoT, and physics-based simulation
TB9: Examon Replication to INFN/BSC Datacentres	The Examon IoTwins-enabled management of testbed N.6 will be replicated on two datacentres of INFN and BSC. The purpose of this use case is to define a methodology for the monitoring infrastructure reuse and deployment in new and different contexts.
TB10: Standardization/Homogenization of Manufacturing Performance	The main objective of this use case is the extension of the models investigated and assessed in testbed N.4 to a wider series of machinery and other plants around the EU.
TB11: Replicability towards Smaller Scale Sport Facilities	This use case will demonstrate the replicability and scalability of testbed N.5 at other FCB sport facilities.

Testbed	Description
Business Oriented	
TB12: Innovative Business Models for IoTwins PaaS in Manufacturing	The testbed aims to validate innovative business-models that bring resources available in the cloud accessible to applications running on manufacturing machines related to the machine monitoring business.

2 IoTwins Platform

The main principles that drive the design of the IoTwins platform are **openness** and **software re-use** respectively. The platform proposes itself as an open framework that manages digital twins in the Cloud-to-Things continuum, built on top of open-source software and tools, and allowing third-party software to integrate by way of open application programming interfaces (API) .

This section provides technical details of the IoTwins platform and lays the foundation for the subsequent platform implementation step. First, we thoroughly discuss the main requirements elicited from test beds in the projects' deliverable document D2.1:" Architecture and technical user requirements ". Second, we present some relevant Unified Modelling Language (UML) use case diagrams aimed at describing the platform operations from the perspective of the three computing layers (IoT, Edge and Cloud). We then propose an architectural design of the IoTwins platform meeting the emerged requirements and accommodating the use cases. Finally, we provide insights on the platform implementation step and describe tools to be reused and integrated for the implementation purpose.

2.1 Requested Capabilities

2.1.1 Data Handling

As one of the Digital Twins' cornerstones, Data Handling becomes extremely important in order to represent (or even forecast) the monitored asset. It is divided into three different consecutive steps: Data gathering, Data transportation and Data storing.
Data Gathering

It is performed by the sensing devices of the testbeds. The variety of sensors involved in the testbeds is quite wide. Thus, the platform has to be able to adapt itself, in order to enable the gathering of different data types and sources, such as:

- Scalar and Primitive types
    - Temperature
    - Acceleration
    - Pressure
    - People counting (turnstiles)

  TB1, TB2, TB3, TB7, TB8 and TB12 specifically require the platform to handle those data types

- IT specific measurement data
    - (Compressed) logs files
    - System and DB Metrics
    - WIFI access point usage logs

TB10, TB5, TB7 and TB12 specifically require to handle this kind of data. However, it is likely that more test beds will gather those metrics in the future, for performing system monitoring.

- Non-IoT Test Bed environment data
    - Binary Files
    - Text Files / JSON
    - Excel of CSV files

  This test beds' internal data is not likely to be gathered by IoT devices, but its gathering is crucial for the Digital Twin accuracy. Some examples could be: Generated executables, sales excel documents, data sheets, etc.

  TB1, TB3, TB5, TB8, TB11 and TB12 will require the handling of this data types.

- Images
  TB5 and TB11 will use IoT devices for taking pictures. However, the handling of this data is limited to the IoT device itself, since only features extracted from the images will be shipped for its transportation, fitting in the Scalar and primitive types, previously explained.

Data Transportation

The protocols used to implement communication in the IoTwins architecture depend on the level (IoT, edge, cloud/backend). The protocols with wider adoption in use for communication with the cloud (from edge to cloud) and within the cloud are OPC UA[1], MQTT[2], and coap[3]. At the IoT level there is a much greater variety of protocols used to attach sensors and smart devices to the edge, e.g. Modbus[4], IEC 60870-5-104[5], Zigbee[6],and proprietary protocols. New technologies like nats.io[7] aim to integrate communication on all levels (IoT, Edge and Cloud) by focusing on the performance requirements.

Currently, data transportation is performed via MQTT (In Test Beds such as TB1, TB5, TB8 or TB11), via Time Series Database direct ingestion (In TBs such as TB2 or TB10), and by shared files plus manual transportation or FTP (in TBs such as TB3 or TB8). The task is, then, two-fold: Enable all required data transportation, but also minimize the number of different technologies used by finding common particularities.

Data Storage

Many different Data Models (D.M) and database solutions are used in the test beds either as a preliminary storage or as long terms storage. Here below is a synthesis:

- Relational DM (TB1, TB8, TB12, ...)
    - PostgreSQL
    - Microsoft SQL Server
- Document DM (TB5, TB11, …)
    - MongoDB, JSON Store

[1] https://opcfoundation.org/about/opc-technologies/opc-ua/
[2] https://mqtt.org/
[3] https://tools.ietf.org/html/rfc7252
[4] https://www.modbus.org/
[5] https://en.wikipedia.org/wiki/IEC_60870
[6] https://en.wikipedia.org/wiki/Zigbee
[7] https://nats.io/

- Time Series DM (TB2, TB3, TB10, …)
    - KairosDB
    - InfluxDB
- Key-Value DM (TB10)
    - Cassandra
- File System (TB1, TB3, TB5, TB8, TB11, TB12, …)
    - Files(Binary, executables,...)
    - CSV Store

It will be important to take into account the link between data models and database solutions, in order to limit the number of database technologies that might share the same data model.

2.1.2 Computation

Computation is requested to be done at all levels: IoT, Edge and Cloud, following real-time, early, and off-line data analysis fashions.

At the IoT level few computation tools may be needed. The tools may be provided either by the data collection tool or as a service of the platform. Light and quick calculation may be done at the IoT level for some types of devices, expect from TB5 and TB11, that require IoT devices with GPU capabilities. Those devices should be able to be orchestrated.

The following operations may be done at the **IoT level**, generically along all the testbeds:

- Filtering
- Data extraction
- Root Mean Square (RMS)
- Minimum, Maximum, Average
- Aggregation
- Format transformation
- Compression

TB1 requests:

- FFT
- Hilbert
- Power Spectral Density (PSD)

- Some other Test Beds require anonymization procedures, such as TB10. Most test beds require encryption at the transportation level, so data will have to be encrypted and decrypted at shipper/producers, and receiver/consumers.
- TB5 and TB11 will also perform feature extraction from rea-time taken photographs.

The following operations may be done at the **Edge level**, generically along all the testbeds:

- Filtering
- Aggregating (w or w/o ELK software)
- Big Data Analytics
- Monitoring dashboard

- ML applications using ML framework (tensorflow, yolo, keras, pytorch)
- Buffering
- Data Conversion
- Searchs

TB5 and TB11 will specifically need:

- Position Triangulating (for WiFi Access Points)
- Processing aggregated data for cameras

Cloud level will be mostly dedicated to off-line computations. Simulations and ML training will be performed at this level. Obviously, all the above listed computations (IoT & EDGE levels) may be performed here. The following lines will list the required capabilities:

- Machine Learning (ML) and AI, generically requested by TBs
    - Anomaly Detection
    - Anomaly Detection Training
    - General ML & AI function to be provided as services
    - Log Analysis (based or not on ELK stack)
- Simulation
    - System simulation with SimulationX (TB1)
    - Machining with Nessy2m (TB8)
    - Agend Based Modelling with Pandora (TB5, TB11)
- Machine Status Fingerprint (TB2)
- Grid Diagnostic Suite (TB7)
- Visualization:
    - Inendi Inspector (TB1)
    - Mineset (TB1)
    - Dashboard(s) (TB1, TB2, TB10)

2.2 Non-functional Security Requirements

2.2.1 Anonymization

Pseudonymisation, which consists of replacing/encrypting/deleting elements directly or almost directly identifying individuals, generally does not protect individuals. Anonymisation of data is a compromise between data and individual protection on the one hand, and the possibility to process data in an interesting way on the other. With anonymization, we seek to address the following three issues[8]:

- Individualization: Is it possible to isolate an individual?
- Correlation: Is it possible to link separate data sets about the same individual?
- Inference: Can information about an individual be inferred from external explanatory variables?

[8] see e.g. Commission nationale de l'informatique et des libertés (CNIL) definition, independent French administrative regulatory body whose mission is to ensure that data privacy law is applied to the collection, storage, and use of personal data, https://www.cnil.fr/fr/lanonymisation-de-donnees-personnelles

The data may be present different levels of risk. Some variables are considered as "identifiers" (the most discriminating variables, e.g. address) or "quasi-identifiers" (when these are cross-referenced, it is possible to isolate an individual), others are sensitive parameters with a high potential value, and finally the residual variables present a priori showing only a low risk of compromise. In order to optimize the compromise, it achieves, anonymization is strongly dependent on three parameters:

- Data typology: the anonymisation methods will be different depending on whether the data are structured (for example, a classic medical database) or unstructured (for example, a social network).
- Data future usage: Depending on the desired studies, an analyst does not need the same information.
- Time dependant: a classical way to attack anonymized data is to use external data to cross with anonymized data and identify the individuals. As the available datasets change with the time, the anonymization process efficiency can change with the time. Some approaches are time-independent

These elements lead to the fact that there is no general anonymization method that applies to any application. Moreover, with anonymization, there is no zero risk to privacy. This risk must therefore be studied and weighed against the possible benefit to be derived from the data.

There are two types of de-identification techniques, which can be combined: those that transform the data to obtain fictitious individuals and those that aggregate the data. Permutation and puzzling techniques change the architecture of the data (e.g. [1] [2]). Noise addition techniques are popular, [3] proposes a state-of-the-art on randomization methods. These methods have the advantage that they do not depend on the history of the data and can therefore be applied during data collection. However, as shown in [4], these methods are easily susceptible to attacks. Differential Privacy (e.g. [5] [6]) is a very popular Privacy notion which involved noise addition. Another way to protect individuals is to create aggregates of individuals. K-Anonymization [7] consists of creating equivalence classes of individuals of minimum class K by generalizing and removing quasi-identifiers, [8] proposes the l-diversity which forces each of the equivalence classes of minimum size K to have different values for the sensitive variables. In the t-proximity, [9] requires that in each equivalence class, the sensitive variable follows the same distribution as in the total population. Generalization and suppression is a complicated NP problem, [10] proposes the use of micro-aggregation to deal with ordinal and continuous categorical variables. The idea here is to partition the data into clusters of minimum size k, then apply an aggregation operator (e.g. the mean for numerical variables) to each cluster and return the aggregated value.

ARX[9] and mu-Argus[10] are open source data anonymization frameworks, which propose both approaches based on generalization (e.g. k-anonymization) and simple differential privacy approaches. Amnesia[11] and mu-ANT[12] are an anonymization framework based on generalization approaches. sdcMicro[13] is a R package that can be used to generate anonymized micro data. K-anonymity is a Python library dedicated to generalization approaches. Diffpriv[14] is a R package which allows some simple differential privacy algorithms. Diffprivlib [15] is Python library developed by IBM for differential privacy. Differential-privacy[16] is C++, Java and Go library proposed by Google to achieve differential privacy for producing aggregate statistics. Thales are developing a Python differential privacy library for time series based on [11] which can be potentially used in

[9] https://arx.deidentifier.org/
[10] https://github.com/sdcTools/muargus/releases/tag/v5.1.3b3
[11] https://amnesia.openaire.eu/
[12] https://github.com/CrisesUrv/microaggregation-based_anonymization_tool
[13] https://cran.r-project.org/package=sdcMicro
[14] https://cran.r-project.org/package=diffpriv
[15] https://pypi.org/project/diffprivlib/
[16] https://github.com/google/differential-privacy

the project. Tensorflow-privacy[17] and PySyft[18] are Python libraries dedicated to achieving differential privacy for Machine Learning. There are also fee-based solutions: aircloak[19], deidentification[20] used for video, Anonymizer[21] for image content.

At the IoT level, two testbeds expressed the need for anonymity in deliverable 2.1: testbeds 10 and 12. At this level, testbed 10 has access to sensitive data from log files. Due to the fact that the data that will be used by TB12 comes from an European automotive OEM, client of MARP, all the data need to be kept under strict confidentiality ruled by an NDA between MARP and the automotive OEM itself so a process of anonymization that prevents from whatsoever association between data, their elaboration and inferenced conclusions is needed before sending them to the Cloud. For TB10, as the choice of the anonymization approaches is use case dependent, the contributor will have to choose among the previously proposed techniques, those that will best meet its requirements.

For TB12, after further investigation on the OEM need for anonymity at IoT level, considering that both the IoT and the Edge are inside the corporate perimeter and the realtime needs of the IoT, it was decided that the anonymization task will be carried out only at Edge level.

At Edge level, testbeds 5/11, 7 and 12 expressed anonymization needs. Testbeds 5/11 will implement anonymization and aggregation techniques directly at the collection point, therefore removing the need for storage or transmission of individual or personal data. Differential Privacy, local differential privacy and incremental generalization can fit the testbeds needs. Testbeds 7 uses energy measurement data of households. For this challenge, smart micro aggregation, differential privacy and data minimization can be used. For Testbeds 12, the stakes are the same that for the IoT level.

2.2.2 Encryption

Encryption requirements were given in the Deliverable 2.1. At the IoT level, encryption may be used as an alternative to anonymization for TB10 and TB12 that operates on sensitive data. All TBs require encrypted channels for data transfers from the IoT level to the cloud.

At the Edge level, multiple computation types will be performed, including aggregation and possibly ML for TB10. It has to be determined if they will be performed on encrypted data provided by the IoT level. All testbeds require encrypted channels from the Edge to the Cloud. For TB2, 3, 5, and 11, a VPN should be provided for data movements from the Edge to the Cloud and vice versa.

At the Cloud level, TBs 2 and 12 require data encryption at rest. With TB12, further investigation is needed in order to clarify if data should be encrypted also during access and whether it should be encrypted in the Edge before being uploaded to the Cloud.

In the following paragraphs, a general description of encryption is given along with its advantages and limitations. It is also explained why anonymization may not always be applied and why encryption is the only viable way to provide confidentiality while dealing with very sensitive data.

[17] https://github.com/tensorflow/privacy
[18] https://github.com/OpenMined/PySyft
[19] https://aircloak.com/
[20] https://www.deidentification.co/
[21] https://www.eyedea.cz/image-data-anonymization/

Encryption converts a plain text into a ciphertext using a secret key. The recovery of the initial information requires the knowledge of the right key, unless someone possesses enough computational resources to verify all of the key possibilities; this is almost impossible as encryption key lengths are chosen in a way that creates a number of key possible values exceeding the attackers' capacities. Encryption is the best way of securing data at motion and one of the ways of securing data at rest.

Regulations, such as GDPR[22], DPA[23], or ePrivacy[24], do not explicitly enforce the use of encryption for data protection. However, they require conducting a Data Protection Impact Assessment (DPIA) whenever the processing could result in a high risk to the data subjects. The DPIA identifies appropriate protection mechanisms, which must be implemented. In the case of confidential data at high risk, encryption is a straightforward choice of protection mechanism, as it is practically the only viable way of providing confidentiality. Alternative techniques – anonymisation, pseudonymisation [12], differential privacy [5] [6], or data fragmentation [13] [14] - can only ensure privacy but not confidentiality. Based on a compromise between utility and privacy, they do not completely eradicate the risk of leakage of some information from the stored dataset. Moreover, the main purpose of anonymisation or differential privacy is to allow the dataset owner to share a transformed, cleaned from sensible information, version of its private data. Therefore, they do not fit a situation, where the data owner wants simply to securely store the data for its own usage and without any information loss.

It should be emphasized that an encryption scheme protects data only as long as the encryption key stays secret. Thus, key management recommendations [15] should be carefully followed. Moreover, it is possible to apply additional methods to protect data powerful attackers able to acquire encryption keys [16] [17] This problem may be tackled in TB10, where questions related to encryption and access control will be discussed.

Two obstacles may discourage from applying encryption: (1) the computational cost of the encryption algorithms leading to a performance slow down and a higher energy consumption, and (2) the difficulty of processing over encrypted data. Efficient solutions exist to deal with the first obstacle. In the Cloud context, the performance overhead coming from encryption is almost negligible, especially while using hardware optimizations such as AES-NI[25]. In environments constituted of highly constrained devices, lightweight encryption schemes are still desirable [18]. Thus, anonymization at the IoT level is the first choice for data protection. However, the successful adoption of standard AES encryption inside many embedded systems raises a debate on the necessity of lightweight cryptography [19].

Enabling efficient processing over encrypted data is one the biggest challenge in the security field. In the project it may be required at the Edge level if the data coming from the IoT level are encrypted. Although fully homomorphic encryption allows to perform calculation over encrypted data without decrypting it first [20] , it is often judged too slow and unpractical. Thus, encrypted databases [26] [27] [28] [29] rely on specialized property-revealing encryption schemes that allow a database server to perform some computations on ciphertexts in response to client queries. Most important of these schemes are order-revealing encryption,

[22] GDPR: European Parliament and Council of European Union (2016) Regulation (EU)
[23] https://www.itgovernance.co.uk/dpa-2018
[24] https://ec.europa.eu/digital-single-market/en/proposal-eprivacy-regulation
[25] https://www.intel.com/content/www/us/en/architecture-and-technology/advanced-encryption-standard-aes/data-protection-aes-general-technology.html
[26] http://www.css.csail.mit.edu/cryptdb/
[27] https://css.csail.mit.edu/mylar/
[28] https://rise.cs.berkeley.edu/projects/arx/
[29] https://www.microsoft.com/en-us/research/project/seabed/

deterministic encryption, and searchable encryption. Based on a compromise between utility and confidentiality, they provide weaker security guarantees than standard encryption schemes [21].

Secure multi-party computation (MPC) seems to be a promising alternative to homomorphic encryption. It allows owners of private datasets to perform operations over their joint data without revealing nothing else but the result of the computation. It is the core component of multiple solutions for machine learning on private datasets, such as tf-encrypted[30] [16], PySyft[31], SCALE/MAMBA[32], or CrypTen[33]. The core idea behind MPC is to split secret data into multiple shares that are spread over a set of independent servers. Computations are then performed in a distributed way. Confidentiality of the inputs is ensured unless an adversary is able to compromise a given threshold of the computing servers (this threshold depends on the adversary model and can be as high as 'all but one' servers).

2.3 IoTwins Use cases

In Section 1.1, requirements elicited from the TBs have been listed. The definition of the use cases presented in this section was indeed guided by those requirements. According to TB's expressed needs, the IoTwins platform will have to exhibit functionalities to:

- Support data transfer to/from the three infrastructure levels in ways that account for the constraints imposed by the specific computation needs (real-time, non-real-time);
- Support several data types and accommodate different data storage needs (short- and long-term storage);
- Support elaboration of data both on the fly (streamed-data elaboration) and at rest (batch-data elaboration)

Apart from offering the users support for building and running services on each of the computing infrastructure levels, the platform will have to also provide its users with services to configure, build and transparently deploy complete "computing chains", i.e., chains of software libraries/tools/services that implement the above-mentioned functionalities along the IoT-Edge-Cloud computing continuum.

In section 2.3.1 and 2.3.2, two different views are presented that describe the usage of the IoTwins platform from the perspective of two different IoTwins stakeholders, the *IoTwins User* and the *IoTwins Developer* respectively. In section 2.3.3, a perspective of the actions undertaken on IoT, Edge and Cloud infrastructures in response to Users' input on the IoT platform is graphically depicted by borrowing the use case notation.

2.3.1 IoT User's use case

The IoTwins User is the main stakeholder that will benefit from the services offered by the IoTwins platform. In the scope of the project, the IoTwins User role is played by Testbeds, who have contributed to identify the requirements of the platform and the cases that we are going to describe. Out of the project scope, potential Users will be players of the manufacturing and facility/infrastructure management sectors that intend to benefit of the IoTwins platform services to implement a digital twin-based management of their business processes. In the Figure 2, the IoTwins User's use case perspective is depicted.

[30] https://tf-encrypted.io/
[31] https://github.com/OpenMined/PySyft
[32] https://homes.esat.kuleuven.be/~nsmart/SCALE/
[33] https://crypten.ai/

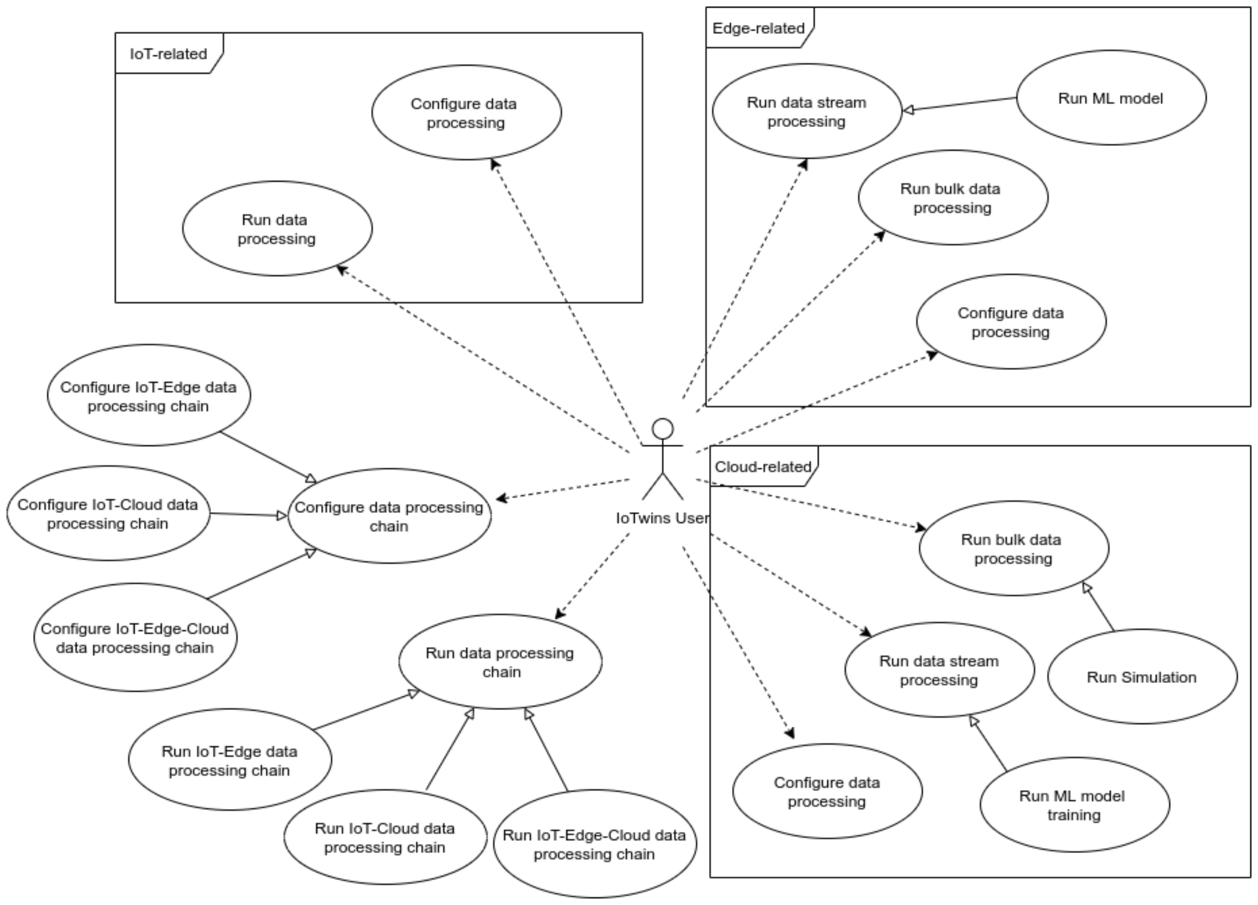

IoT-related
Configure data processing
Run data processing
Edge-related
Run data stream processing
Run ML model
Run bulk data processing
Configure data processing
Configure IoT-Edge data processing chain
Configure IoT-Cloud data processing chain
Configure data processing chain
Configure IoT-Edge-Cloud data processing chain
IoTwins User
Cloud-related
Run bulk data processing
Run data stream processing
Run Simulation
Run data processing chain
Run IoT-Edge data processing chain
Run IoT-Cloud data processing chain
Run IoT-Edge-Cloud data processing chain
Configure data processing
Run ML model training

Figure 2: IoTwins User's use case diagram

The IoTwins platform will basically offer Users services to **configure** and **run** data processing tasks on IoT, Edge and Cloud respectively. Under the "data processing" umbrella, all activities and tasks pertaining to data analytics (data filtering, data polishing, data integration, data elaboration, data visualization, data monitoring) are covered. In the Figure 2, use cases are categorized according to the infrastructure where computing tasks are run. Furthermore, the platform will provide the User with services to build "data processing chains", i.e., a collection multiples services, distributed on at least two different infrastructures, chained together to serve the specific computing needs. A short description of the depicted use cases follows hereafter.

The category "IoT-related use cases" includes two use cases: *Configure data processing* and *Run data processing*. These refer to the action of configuring and running computing tasks on data sensed by sensors that the IoT device is equipped with. Depending on the need and the capabilities of the device, computing tasks may consume data on-the-fly or data at rest. Given the constraints of that characterize IoT devices, computing tasks expected to run are of data preparation type (e.g., data cleaning, data pre-filtering, etc.)

"Edge-related use cases" includes the following use cases: *Configure data processing, Run bulk-data processing, Run data stream processing and Run ML Model*. *Configure data processing* allows the User to set-up an environment for data elaboration. Configuration involves actions like a) setting up data sources address, type, format, b) choosing and setting up the parameters of a specific computing task, c) selecting the destination of the data output, etc. *Run bulk-data* is the action of invoking tasks that elaborate data at rest, i.e., data locally stored, while *Run data stream processing* refers to the elaboration of live data coming from IoT devices. Finally, the *Run ML model* is a sample data stream processing use case.

"Cloud-related use cases" includes the following use cases: *Configure data processing, Run bulk-data processing, Run data stream processing, Run ML training* and *Run Simulation.* Basically, most of use cases available in the Edge are available in the Cloud too. The User can configure and run computing tasks on data streams coming from either IoT or Edge, or on data-at-rest stored in the Cloud. For instance, the User can request to run Simulations (*Run Simulation* use case) or to train ML models (*Run ML training* case) on bulk data residing in the Cloud. Trained ML models can then be moved to the Edge where, fed by data streams coming from IoT devices, will execute (see Edge's *Run ML Model* use case).

Finally, use cases are have been defined to let the User request the activation of multiple tasks distributed along the chain of the computing infrastructures. The User can set-up a data processing chain (*Configure data processing chain* use case) and run it (*Run data processing chain* use case). Depending on the needs, chains can be configured to span two infrastructures (IoT - Cloud, IoT - Edge) or all infrastructure levels (IoT-Edge-Cloud). As an explanatory example, a typical data processing chain envisions data sensing and filtering tasks deployed in the IoT, an ML model training task running in the Cloud and fed with data at rest, and an ML model execution task running in the Edge consuming IoT data streams. Along the chain, data transfer services (bulk data transfer, live data transfer) and storage services (Relational/NoSQL/Time series DB, file-system based storage, etc.) can be instantiated (*Configure data processing chain* use case).

2.3.2 IoT Service Developer's use case

The IoTwins Service Developer is the actor in charge of developing IoTwins services, to be included in the repository of IoTwins services, which IoTwins Users may take advantage of. One of IoTwins project's goal is to release open API and deliver guidelines that the Service Developer will use to implement such services. In the Figure 3, the IoTwins Service Developer's use case perspective is depicted.

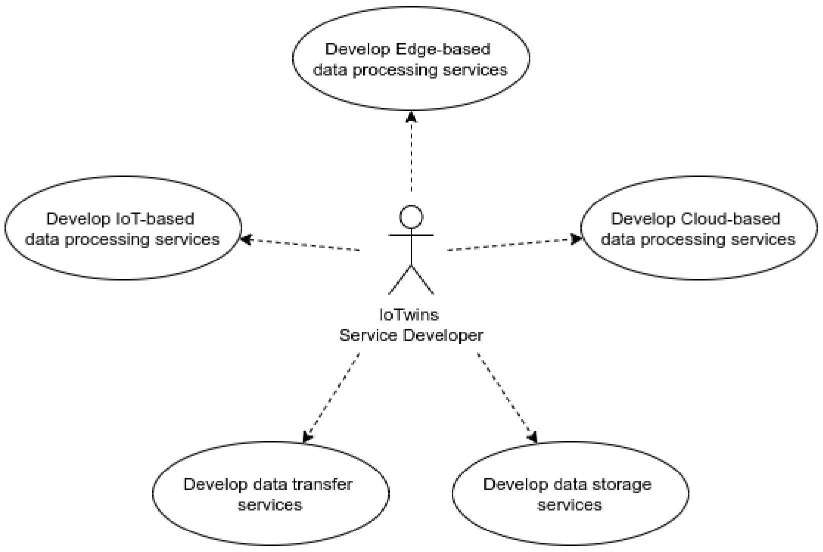

Develop Edge-based data processing services
Develop IoT-based data processing services
Develop Cloud-based data processing services
IoTwins Service Developer
Develop data transfer services
Develop data storage services

Figure 3: IoTwins Service Developer's use case diagram

The IoTwins platform offers the Service Developer APIs to develop both data processing services and data storage/transfer services that are requested to build data processing chains. When developing processing services, the Service Developer must also implement service configuration tools, so to allow the User to configure data sources/types/formats, as well as all parameters requested by the processing tasks. API will be differentiated for the development of tasks according to the hosting infrastructure, i.e., IoT, Edge or Cloud (*Develop IoT-Based data processing services* use case, *Develop Edge-Based data processing services* use case and *Develop Cloud-Based data processing services* use case respectively). Finally, the Developer is requested to develop data transfer and data storage services that will have to support data processing along the chain (*Develop data transfer services* use case and *Develop data storage services* use case).

2.3.3 Computing infrastructures use cases

The purpose of the following diagram is to present a view that identifies all potential actions carried out at each infrastructure level where computation may occur (IoT, Edge and Cloud respectively). Considering this specific objective, we are not interested in showing what the end user's point of view may be in respect to the platform's front end offered functionality (for that viewpoint, the reader may refer to IoTwins User's perspective described earlier in this section). We will rather focus on which actions may be triggered at each level of computation in consequence of prior end-user's choices. This perspective will drive the design of IoTwins platform functionality on a per-level basis. In the proposed use case view, we re-interpret the "actor" concept of the UML notation and give the "devices" that populate the three computing levels the role of action initiators.

In Figure 4, the overall use case diagram is depicted. Basically, all cases can be classified into three distinct categories:

- Data Storage-related use cases. To this category belong all use cases that involve the storage of data locally on the considered device
- Data Transfer-related use cases. This category collects use cases addressing the need for transfer of data (be it inbound or outbound the considered device)
- Data Elaboration-related use cases. It collects use cases that concern local elaboration of data

In the picture shown in Figure 4, each category is assigned a colour: light orange for data storage, light blue for data transfer and light green for data elaboration.

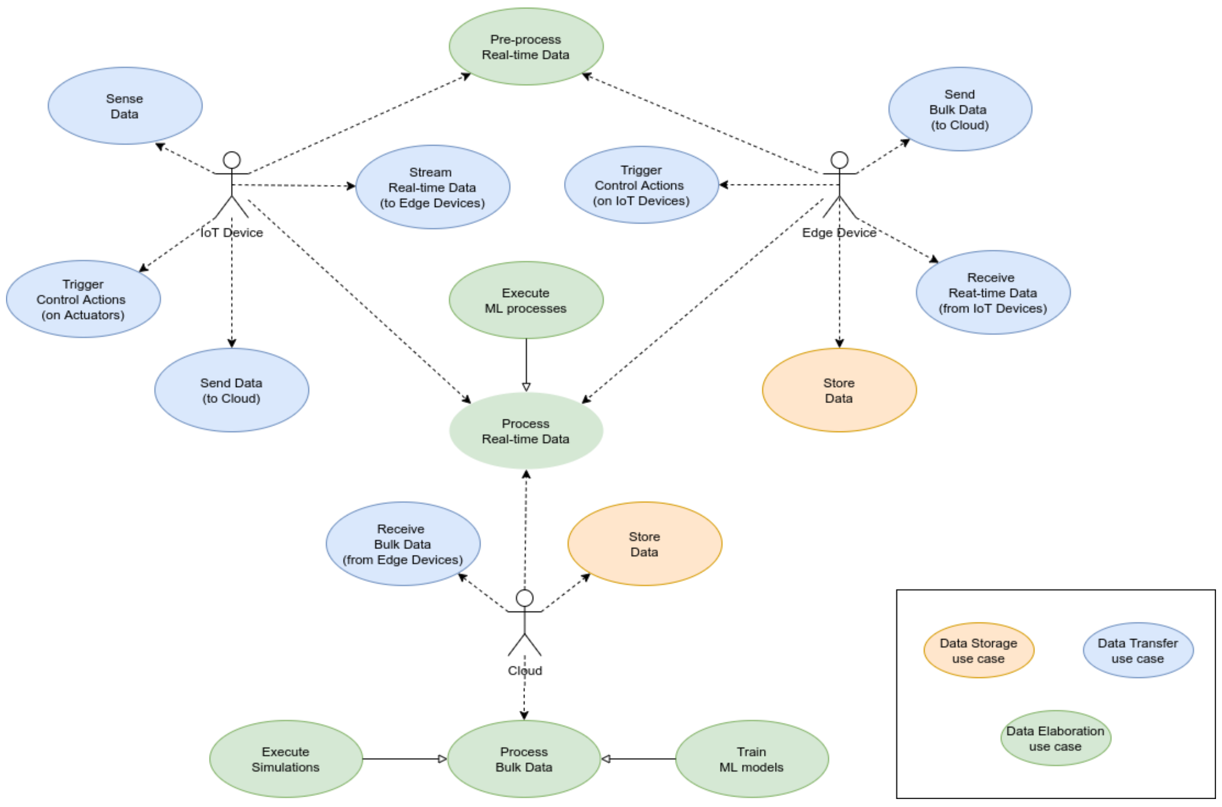

Pre-process Real-time Data
Sense Data
Send Bulk Data (to Cloud)
Stream Real-time Data (to Edge Devices)
Trigger Control Actions (on IoT Devices)
IoT Device
Edge Device
Trigger Control Actions (on Actuators)
Execute ML processes
Receive Real-time Data (from IoT Devices)
Send Data (to Cloud)
Store Data
Process Real-time Data
Receive Bulk Data (from Edge Devices)
Store Data
Cloud
Execute Simulations
Process Bulk Data
Train ML models
Data Storage use case
Data Transfer use case
Data Elaboration use case

Figure 4: IoTwins use case diagrams from the computing infrastructure perspective

In the following, a high-level explanation of all triggerable actions is given for each computing infrastructure level.

2.3.3.1 *IoT device*

- Sense Data. It is the basic functionality of gathering real-time data from attached sensors
- Trigger control action (on actuators). This action refers to the functionality of sending a control command to the attached actuator devices
- Stream (real-time) data (to Edge device). This action consists in uninterruptedly, sequentially and timely transfer of sensed data to the Edge device
- Send data (to Cloud device). This action consists in sending to the Cloud device sensed (or locally memorized) data subjected to no real-time constraint (short-lived transfer)
- Pre-process data. This action consists in pre-processing sensed data before sending it to the Edge device (e.g., data filtering, polishing, etc.)
- Execute ML processes. To predict future data, detect anomalies, or optimize control in general, minimal machine learning processes can already be executed on the IoT level, in case the hardware permits it.

2.3.3.2 *Edge device*

- Send bulk data (to Cloud). It refers to the action of sending (locally stored) bulk data to the Cloud (long-lived transfer, no real-time constraint)
- Trigger control action (on IoT device). It is the action of sending a control command to the IoT device. The control command is the output of some data elaboration performed locally
- Receive data (from IoT device). It refers to the action of receiving data from the IoT device in both real-time and non-real-time fashion
- Pre-process data. It is the action of running some data processing over data coming from the IoT device
- Store data. The action of locally storing data received from the IoT device for either later elaboration or transfer to the Cloud and local processing by the ML processes. In case these processes need data that is only available on the cloud level (e.g. weather predictions), the data storage on the edge device might also store data received from the cloud level
- Execute ML processes. To predict future data, detect anomalies, or optimize control in general. These processes may directly access the stream of data received from (locally attached or IoT level) sensors or processes historical data from the local data storage. The execution of these models on the Edge level may need additional data from the cloud level (which in turn could be streaming data or bulk data from the local storage)

2.3.3.3 *Cloud*

- Receive bulk data (from Edge devices). It is the action of receiving bulk data sent by the Edge device (long-lived transfer, no real-time constraint)
- Store data. The action of locally storing data received from IoT and Edge devices for later elaboration
- Process streamed data. It is the action of running some data processing over data while they come from the Edge device
    - Execute ML processes. To predict future data, detect anomalies, or optimize control in general. These processes may directly access the stream of data received from/via the Edge devices or processes historical data from the cloud level data storage.

- Process Bulk data. The action of running some data processing over data locally stored simulations)
    - Train ML models. It concerns the training of ML models on locally stored data. ML models can either execute in the Cloud or be sent to the Edge
    - Execute simulation tools. Simulations tools can be set up for several purpose. Basically, simulators access data from the cloud level data storage

2.4 IoTwins architecture

In this section, we discuss the IoTwins architectural design that fulfils the functional needs emerged in the definition of the IoTwins use cases.

In Figure 5, the three involved computing infrastructure layers are depicted along with the available assets.

On the IoT side, Sensors, Smart Sensors, Actuators and IoT Gateways represent the typical assets available for sensing data from the environment and sending control signals respectively. IoT Gateways are responsible for providing a unique point of access to Sensors/Actuators. Computing capacity is scarce and can be exploited to make data pre-processing (e.g., filtering).

On the Edge side, computing assets such as commercial off-the-shelf (COTS) PC or industrial PC (IPC) are usually available, but more powerful units can sometimes be employed. Those provide enough computing capacity to run more sophisticated data elaboration routines.

On the Cloud side, data centres and HPC offer powerful computing nodes in an aggregated fashion. Parallel and distributed computing paradigms can be exploited to perform computing intensive tasks.

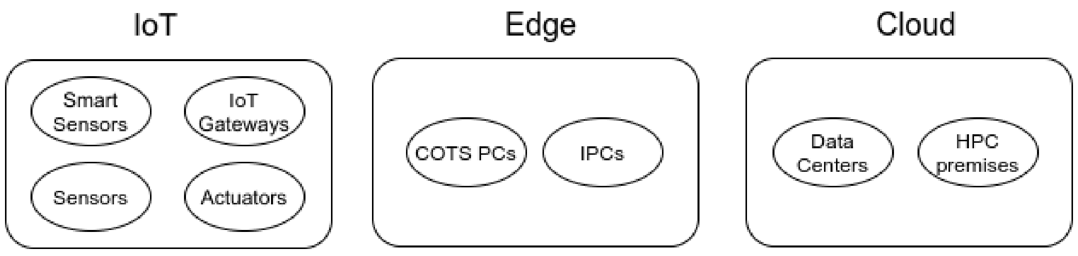


Figure 5: IoTwins Asset layer

In Figure 6, we have depicted a layered view of the IoTwins architecture that will have to serve the just described computing environment. The four-layered IoTwins architecture will exploit and manage computing assets to accommodate end-users need.

While the computing context spans three different levels (IoT, Edge and Cloud respectively), the purpose of the IoTwins platform is to build out of the three levels a *continuum* of computing resources that is able to address in a flexible way the requirements of all testbeds. Despite available computing resources are geographically distributed, the IoTwins architecture strive to follow a centralized approach, i.e., the fraction of the architecture components expected to run on the computing resources to be managed is extremely low.

Following the principle of software re-use, many off-the-shelf and partner-owned open source software are going to be employed as official components of the IoTwins architectures. Still, the architectural design is intentionally left "open" in a way that allows other third-party solutions to be integrated with little development efforts.

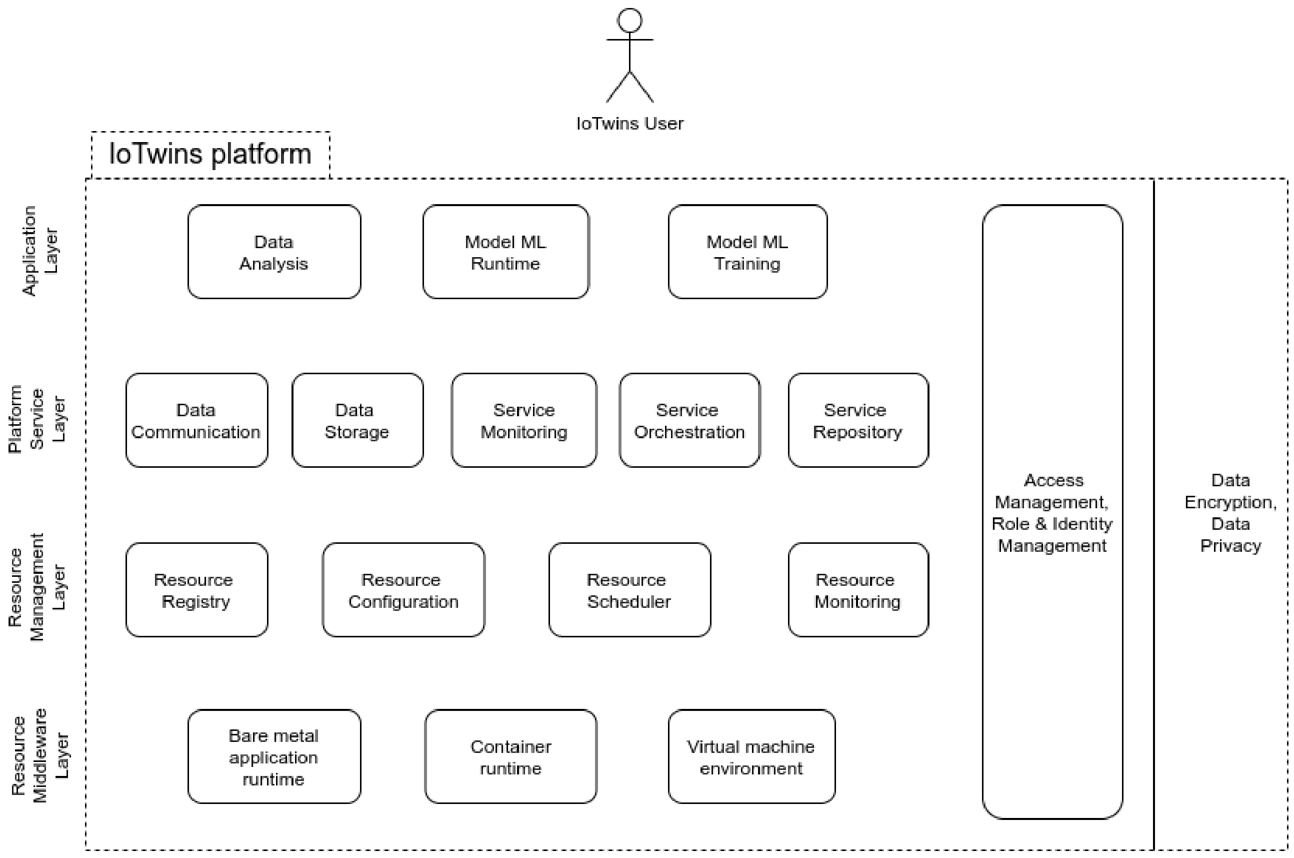


Figure 6: IoTwins Architecture

2.4.1 Resource Middleware Layer

The Resource Middleware layer is populated by tools, software components and libraries that implement the virtualization of the underlying computing assets and provide high level functionality to manage them as high-level resources. By the term "resource", we will refer to a virtualized computing resource (virtual machine, virtual storage, container, etc.) ready to use for general-purpose computation.

On the Edge and IoT level, multiple virtualization and encapsulation alternatives are available. Apart from the Docker runtime there are several alternatives that can be utilized on resource limited devices at both the edge and IoT level so that Open Containers Initiative[34] (OCI) compatible containers can be used to uniformly distribute applications for sensing, controlling, and data filtering to a variety of devices by including the execution environment (libraries, binaries, etc.) as needed.

There are different possibilities to add remote application management functionality to the edge and IoT devices. The remote management components on the edge and IoT devices are closely connected to the functionality provided on the Service Provisioning Layer. KubeEdge for example (see section 4.3.1.2) is a lightweight solution to enable remote management of containerized applications by a Kubernetes based backend (Kubernetes integration with OpenStack is possible using the Magnum component[35], Mesos[36] together with the Marathon framework[37].

[34] https://opencontainers.org/
[35] https://wiki.openstack.org/wiki/Magnum
[36] http://mesos.apache.org
[37] https://mesosphere.github.io/marathon/

2.4.2 Resource Management Layer

This layer provides the upper layer with the functionality to manage Resources (I.e., virtualized computing resources) in a transparent way. In particular, three components populate this layer:

- Resource registry. It keeps track of all resources, their features, their availability, etc. Whenever a new resource is acquired by the platform, a new entry (record) representing that resource will be added to the registry
- Resource scheduler. Upon the arrival of a resource request, it is responsible for searching the computing resource(s) that fit and schedule its activation on behalf of the requestor
- Resource monitor. It implements the monitoring of all active computing resources. Resource level utilization will be monitored through a set of KPIs

2.4.3 Platform Services Layer

The Service provisioning layer is responsible for the provisioning of IoTwins “services”. The basic idea is that services are implemented as OCI compliant containers and their images are stored in a container image registry accessible from any computing unit. Most of IoTwins services will be pre-packaged and pre-loaded on the registry. The system will allow for packaging and upload of customer services (typically, custom applications for data elaboration such as data filtering, simulators, ML models, etc.).

A service can belong to one of the following types:

- Data transfer. This category collects all services that offer support for on-to-one, one-to-many and many-to-many communication. Support will have to be offered for real-time, bulk data, short-lived and long-lived transfer in general. Example of services providing data transfer support are message brokers based on the publish/subscribe pattern (e.g., Kafka), FTP, NTP, webdav, etc.
- Data storage. Data storage services take care of the need of storing data locally to the computing unit where the service has been requested. Data storage services will include relational databases, NOSQL databases, Time series DB, file system-based storage, etc...

The Service Monitoring component is in charge of monitoring the provisioned services and informing the upper layer about their performance (e.g., end-to-end delay in the case of a data transfer, storage capacity saturation in the case of a data storage service, etc.)

In this layer, the Service Orchestrator component is responsible for carrying out the orchestration of IoTwins applications. In particular, the Orchestrator is in charge of managing the life-cycle of applications, starting with the provisioning of all required software components and ending with their disposal. It will monitor applications’ health (in terms of QoS) and take corrective actions in case the QoS level is not being sustained as requested.

Concerning the provisioning task, upon the arrival of an end-user request the Orchestrator will a) schedule resources in the computing continuum (IoT -> Edge -> Cloud), b) retrieve the tools/software components/libraries useful to build up the distributed application, c) install the software components on the target devices, do the necessary pipelining (configuring and adequately connecting components to each other), d) set up and run the application monitoring infrastructure and d) run the application.

2.4.4 Application Layer

This layer is populated by customer applications that perform some kind of elaboration over data. On the IoT side, it can be an application that filters or polishes data before sending it to the Edge. On the Edge side, it can be an ML model that takes in input data coming from IoT and elaborates actions (e.g., triggering commands to actuators). On the Cloud side, it can be a simulator that works on bulk data provided by the Edge or the training of an ML model.

2.4.5 Access Management, Role and Identity management

Identity management or identity and access management (IAM) is a framework of policies and technologies for ensuring that the proper users access the appropriate resources in a computing infrastructure. Identity and access management systems not only identify, authenticate, and authorize individuals but also applications and services that can access certain resources.

An IAM service provides the following functions:

- **Authentication**: to verify the users/application identity
- **Enrolment**: provides enrolment and registration functionalities so that users can join groups/collaborations according to well-defined flows.
- **Attribute and identity management**: provides services to manage group membership, attributes assignment and account linking functionality.
- **User provisioning**: the IAM provides endpoints to provision information about user's identities to other services, so that consistent local account provisioning, for example, can be implemented.

A possible solution to implement the Access Management Layer is the one provided by the INDIGO Identity and Access Management Service (see section 4.1.1)

3 IoTwins Architecture Details

The following sub-sections present the internal structure of the three Architectural Levels and possible solutions that can be used to implement the described components or functionality.
Figure 7 shows an overview of the three levels of the IoTwins Architecture and how they are interconnected:

- The Cloud Level provides components for central resource management of all architectural levels, which means that it controls which software is running on which hardware (orchestration) in the complete system. In addition, the Cloud Level stores and provides data from and to the devices on the Edge Level. Third, the Cloud Level provides services to process the data and create (partial) models of the system that can be used on Cloud Level, Edge Level and/or IoT Level.

- The Edge Level provides components that implement the central orchestration decisions on the edge devices. The orchestration functionality on the Edge Level is also used to run the models that were created on Cloud Level. In addition, the Edge Level devices store and forward sensor data to the Cloud Level and retrieve data from the Cloud Level.

The IoT Level is the most diverse Level. It comprises integrated smart devices that implement all necessary connectivity and pre-processing functionality as well as devices that in their architecture resemble the Edge Level and thus provide methods to orchestrate functionality on these devices.

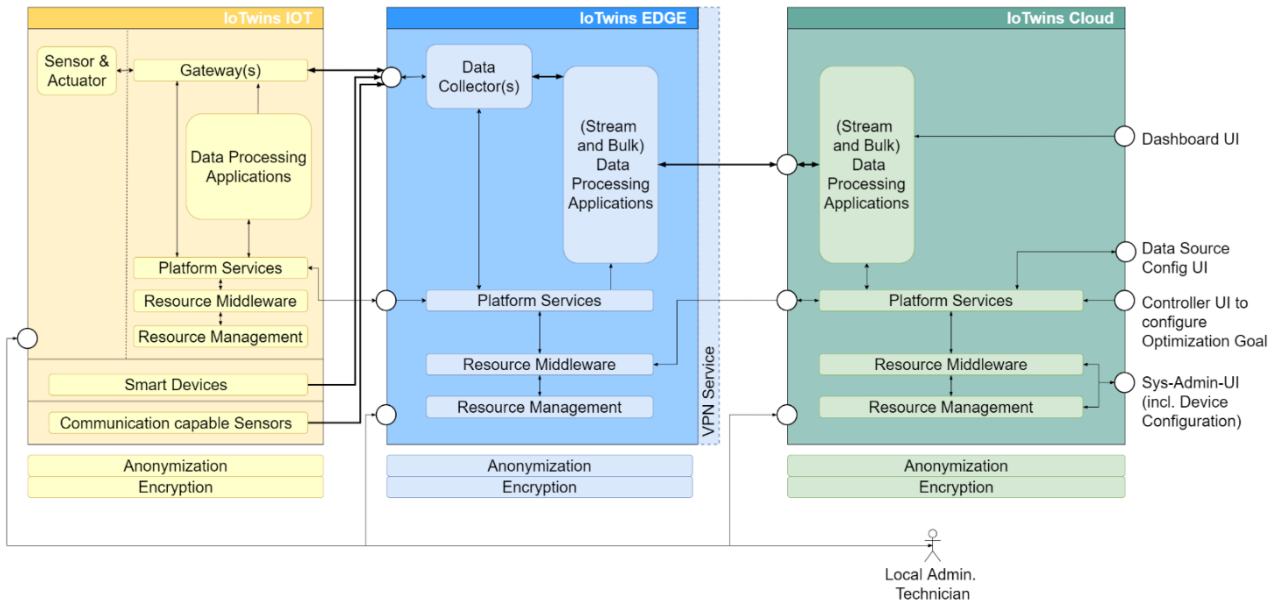


Figure 7: IoTwins Architecture Implementation Overview

3.1 IoT Level

This section is meant to describe the function blocks of the IoT layer and describe the solutions already in place at testbeds. The list of function blocks is based on the current work in T2.2: “Platform Architecture Design and API specification” and non-exhaustive.

The IoT level of the different testbeds is, of course, highly heterogeneous, therefore different approaches were considered when defining the function blocks for the IoT Level.

Nevertheless, the function blocks can be generalized to cover following areas:

- **Data Collection**

  The collected data is expected to be fairly different in terms of format. These real-time measurements go from temperature/acceleration/pressure/… data to text-log metrics collections or even to real-time photography, from communication capable sensors (TB5). Data collected at IoT level can be processed on-site or after its shipment.

- **Computation**

  The IoT level comprises a wide range of different computational techniques. Concretely, its use cases are divided into two groups: Analysis and detection, and data preparation.

  In the first group, **analysis and detection**, data is processed in a real-time basis, even before its shipment to the following layers, in order to be able to detect or forecast anomalous behaviours that should trigger an action.

  This is done, mainly, by executing Machine Learning models against the real-time data. TB2, TB5, TB11 are some Test Best in which this need is expected. ML Model(s) box in Figure 8 represents this behaviour.

  Another way of performing safety-techniques is by checking whether the data is within an already set limits or ranges, by performing checks and comparisons. An example of this supervision technique use case is TB12.

In the second group, **data preparation**, data is manipulated in order to be pre-processed before its shipment. This typically includes aggregation, conversion, filtering or compression. Data preprocessors box in Figure 8 represents this behaviour.

Testbeds are expected to benefit from any (or both, consecutively) of these computational techniques at the edge level. However, the edge should be prepared to accept (or flexible enough to set) different environments, such as windows exe programs (As requested by TB8).

- **Data Transport**

  Although the collected data come from multiple sources (e.g. cameras, engines, turnstiles), the data to be transported typically follows a more-standard fashion, thanks to the computation block. For example, complex data, such as photographs, is expected to be pre-processed before its transportation to the node. In consequence, transported data will typically be lighter, with formats such as CSV files, text, key-values, or binary files (TB12). Its shipment should be done with MQTT/HTTPS/REST API/Rsync as requested by TBs

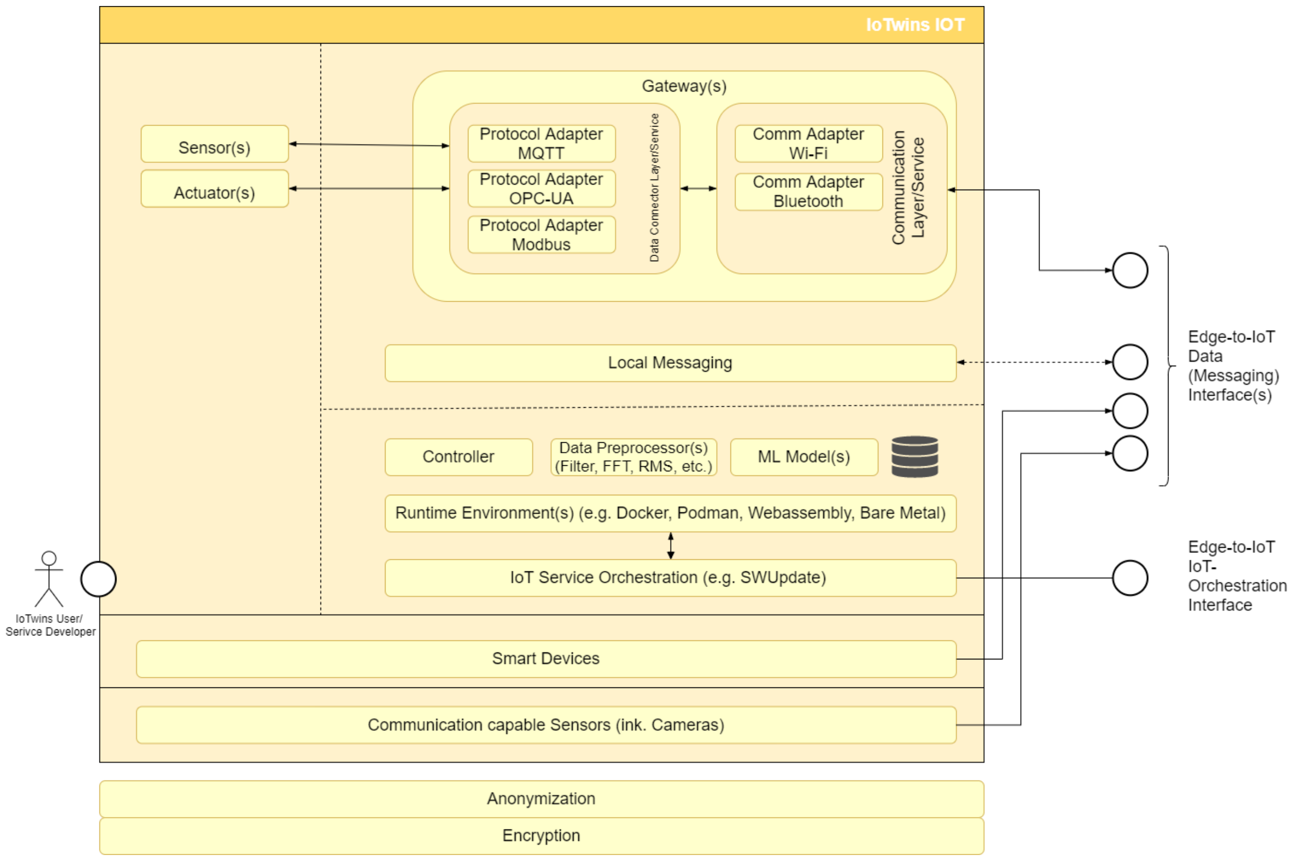

IoTwins IOT
Gateway(s)
Sensor(s)
Actuator(s)
Protocol Adapter MQTT
Protocol Adapter OPC-UA
Protocol Adapter Modbus
Data Connector Layer/Service
Comm Adapter Wi-Fi
Comm Adapter Bluetooth
Communication Layer/Service
Local Messaging
Controller
Data Preprocessor(s) (Filter, FFT, RMS, etc.)
ML Model(s)
Runtime Environment(s) (e.g. Docker, Podman, Webassembly, Bare Metal)
IoT Service Orchestration (e.g. SWUpdate)
Smart Devices
Communication capable Sensors (ink. Cameras)
Anonymization
Encryption
IoTwins User/ Serivce Developer
Edge-to-IoT Data (Messaging) Interface(s)
Edge-to-IoT IoT-Orchestration Interface

Figure 8: IoTwins Architecture Implementation IoT Level

3.2 Edge Level

On the edge level the IoTwins architecture provides functionality for

- **Sensor data collection and storage**: The connection of sensors to the edge devices is highly dependent on the test bed setup. Thus, adapters provide mapping functionality for streaming data and for historical data (for batch processing). In Figure 9, these functionalities are found in "Data Collector(s)", "Local Database", "Global Information Retrieval", "Long Term Data Upload"

- **Data Processing**: In WP3: "AI Services for distributed digital twins", services are developed that analyse and predict sensor data (see D3.2: "First version of the simulation and AI services for digital twins"). This functionality is provided in OCI containers that can be managed using the "Edge Service Orchestration". In Figure 9 these services are found in "Data Processor(s)", "ML-Model(s)", "ML Model Quality Notification", "Controller(s)"

- **Management of services provided on the edge level**: The edge level provides methods to manage the provided functionality (apps) dynamically. In the IoTwins architecture, the applications are packed into OCI containers. The Edge Service Orchestration thus has to provide functionality to download, start, stop and configure OCI containers. In sub-section 4.3.2 methods are described on how services that are provided using OCI containers can be managed on resource restricted, not always connected devices. In Figure 9 see "Edge Service Orchestration"

- **Management of functionality on the IoT level**: In case one of the IoT Level devices, which are connected to the Edge device, allows orchestration of services on the device, the Edge device has to be enabled to forward commands to the IoT Level. This functionality depends on the possibilities and interfaces offered by the IoT Level device. One example for such a system that can be used to update IoT Level Devices is SWUpdate (see also section 4.2.1), for which a proxy could be started (temporarily) on the Edge devices. In Figure 9 this is illustrated as "IoT Level Control Logic"

- **Communication with both the IoT level and the Cloud level**: "Data Collector(s)", "Global Information Retrieval", and "Long Term Data Upload" in Figure 9.

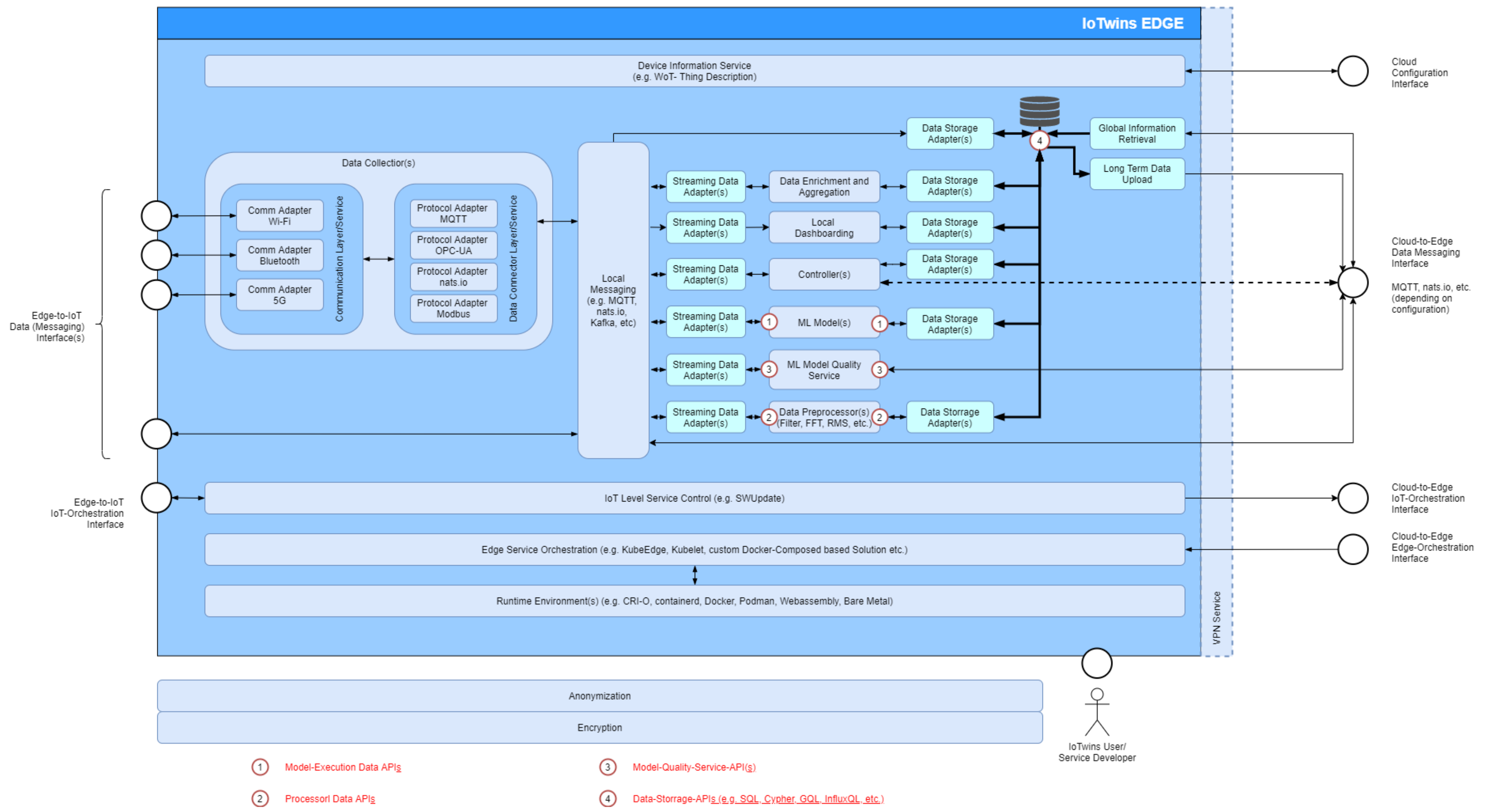

IoTwins EDGE
Device Information Service (e.g. WoT- Thing Description)
Cloud Configuration Interface
Data Storage Adapter(s)
Global Information Retrieval
Long Term Data Upload
4
Data Collector(s)
Comm Adapter Wi-Fi
Comm Adapter Bluetooth
Comm Adapter 5G
Communication Layer/Service
Protocol Adapter MQTT
Protocol Adapter OPC-UA
Protocol Adapter nats.io
Protocol Adapter Modbus
Data Connector Layer/Service
Local Messaging (e.g. MQTT, nats.io, Kafka, etc)
Streaming Data Adapter(s)
Data Enrichment and Aggregation
Local Dashboarding
Controller(s)
ML Model(s)
ML Model Quality Service
Data Preprocessor(s) (Filter, FFT, RMS, etc.)
Data Storrage Adapter(s)
1
2
3
Edge-to-IoT Data (Messaging) Interface(s)
Cloud-to-Edge Data Messaging Interface
MQTT, nats.io, etc. (depending on configuration)
Edge-to-IoT IoT-Orchestration Interface
IoT Level Service Control (e.g. SWUpdate)
Cloud-to-Edge IoT-Orchestration Interface
Edge Service Orchestration (e.g. KubeEdge, Kubelet, custom Docker-Composed based Solution etc.)
Cloud-to-Edge Edge-Orchestration Interface
Runtime Environment(s) (e.g. CRI-O, containerd, Docker, Podman, Webassembly, Bare Metal)
VPN Service
Anonymization
Encryption
IoTwins User/ Service Developer
1 Model-Execution Data APIs
2 Processorl Data APIs
3 Model-Quality-Service-API(s)
4 Data-Storrage-APIs (e.g. SQL, Cypher, GQL, InfluxQL, etc.)

Figure 9: IoTwins Architecture Implementation Edge Level

3.3 Cloud Level

On cloud level the architectural function blocks are:

- **Data communication and management to/from Edge devices:** Functional block that provides services for the data exchange between the Edge and Cloud layers of the architecture. It’s composed by a data transfer plane and a management plane. The data transfer performs the actual data movements using the protocol requested by the testbeds (i.e. S3, HTTPS, WEBDAV, SSH, MQTT). The management plane performs user authentication/authorization, error identifications and integrity checks.

- **Service Orchestration:** It's responsible for automating the deployment of applications, provides mechanisms that deploy, maintain, and scale applications-based rules, metrics and dependencies among the applications themselves.

- **Resource Management:** Enable testbed operators/users to deploy and provision cloud resources on demand, created from cloud templates. This layer manages the storage, computing and network resources. It provisions virtual server, associates some storage to that server and set up some network and security configurations regardless of the underlying architecture. GPUs, if available, can be associated too. It exposes management interfaces via GUIs or APIs. APIs can be exploited by higher level orchestrating services, i.e. multi-cloud PaaS orchestrators

- **Data Processing:** Data can be processed by running applications directly on virtual machines instantiated by the resource manager or within containers shipped to the VMs via a service orchestrator. GPUs and other accelerators, if made available to the VMs, can be exploited to accelerate the computations. Applications are shipped by the testbed in collaboration with WP3 and can be roughly categorized in predictive services that requires artificial intelligence frameworks and simulations that, in some cases, require interactive graphical access, GUIs and 3D rendering visualization.

- **Data Access:** Storage resources will be allocated to the testbeds in dedicated spaces. Moreover, the roles of Data Owner, Data Manager, Data Operator have been identified with decreasing rights on the data. The data access policies will respect these roles. Regarding Storage Quality of Services, no strong requirements emerged from the TB interviews. Few testbeds (TB5 and TB11) requested a lower limit on the performance (in terms of IOPS) of the storage systems in the order of what is provided by standard SSD disks. Replication capabilities will be available in order to guarantee data safety.

- **Database/DataStore for different Data Models:** According to the testbeds requirements, several technologies and services could be provided to support different Data Models (i.e. Document store, key-value, time-series, relational databases) and connect those resources to the data processing pipelines provided by WP3.

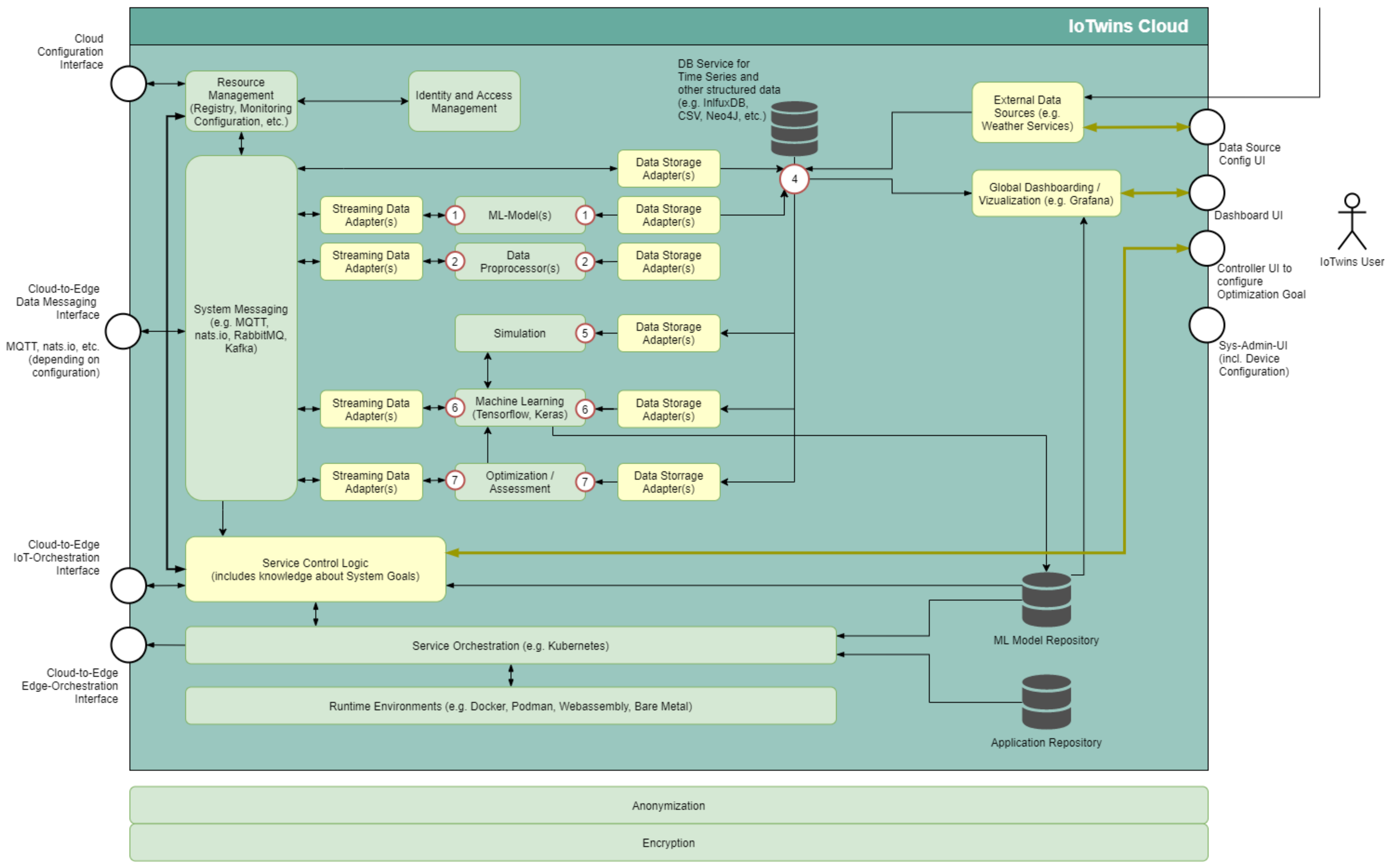

IoTwins Cloud
Cloud Configuration Interface
Resource Management (Registry, Monitoring Configuration, etc.)
Identity and Access Management
DB Service for Time Series and other structured data (e.g. InlfuxDB, CSV, Neo4J, etc.)
External Data Sources (e.g. Weather Services)
Data Source Config UI
Global Dashboarding / Vizualization (e.g. Grafana)
Dashboard UI
IoTwins User
Controller UI to configure Optimization Goal
Sys-Admin-UI (incl. Device Configuration)
Cloud-to-Edge Data Messaging Interface
MQTT, nats.io, etc. (depending on configuration)
System Messaging (e.g. MQTT, nats.io, RabbitMQ, Kafka)
Streaming Data Adapter(s)
ML-Model(s)
Data Storage Adapter(s)
Data Proprocessor(s)
Simulation
Machine Learning (Tensorflow, Keras)
Optimization / Assessment
1
2
4
5
6
7
Cloud-to-Edge IoT-Orchestration Interface
Service Control Logic (includes knowledge about System Goals)
ML Model Repository
Service Orchestration (e.g. Kubernetes)
Cloud-to-Edge Edge-Orchestration Interface
Runtime Environments (e.g. Docker, Podman, Webassembly, Bare Metal)
Application Repository
Anonymization
Encryption

Figure 10: IoTwins Architecture Implementation Cloud Level

4 IoTwins Software Stack

This section summarizes the tools and services available to fulfill the functions described prior.

4.1 Comprehensive Tools

4.1.1 Identity and Access Management Service

A tool proposed for inclusion in the IoTwins infrastructure is the IAM service developed by the INDIGO-DataCloud EC project[38] and maintained for the foreseeable future by the INFN partner. The INDIGO Identity and Access Management Service[39] is developed as an opensource software that provides a layer where identities, enrolment, group membership and other attributes and authorization policies on distributed resources can be managed in a homogeneous way, supporting identity federations and other authentication mechanisms (X.509 certificates and social logins). Al the main functionalities described above are implemented by the INDIGO-IAM that supports, as authentication methods, SAML[40] Identity Providers (IdPs) or identity federations, OpenID Connect providers and X.509 certificates.

The INDIGO-IAM service has been successfully integrated with many off-the-shelf components like Openstack, Kubernetes, Atlassian JIRA and Confluence, Grafana and several middleware services that can be used in the IoTwins architecture.

INDIGO-IAM is based on the OpenIDConnect and OAuth2.0 protocols and supports two types of users:

- **Normal users**: these users can login with the IAM, register client applications that use the IAM for authentication, link external accounts to their account (when allowed by the configuration)
- **Administrators**: these are users that have also administration privileges for an IAM organization (i.e. can manage groups, create users, etc.)

The service exposes the OpenID Connect/OAuth dynamic client registration functionality offered by the MitreID OpenID Connect server libraries. In OAuth terminology, a client is an application or service that can interact with an authorisation server for authentication/authorization purposes and a new client can be registered in the IAM in two ways:

- Using the dynamic client registration API;
- Via the IAM dashboard (which simply acts as a client to the API mentioned above).

After registration the client can obtain a token using APIs or already available scripts and GUIs.

[38] https://www.indigo-datacloud.eu/
[39] https://indigo-iam.github.io/docs/v/current/
[40] https://wiki.oasis-open.org/security/FrontPage

4.2 IoT Level Software

4.2.1 IoT Level Service Control

SWUpdate[41] is a Linux Update agent with the goal to provide an efficient and safe way to update an embedded system. It supports the common media on embedded devices such as NOR / NAND flashes, UBI volumes, SD / eMMC, and can be easily extended to introduce project specific update procedures. Here a short list of the main features:

- Install on embedded media (eMMC, SD, Raw NAND, NOR and SPI-NOR flashes)
- Allow delivery single image for multiple devices
- Multiple interfaces for getting software
  - Local storage
  - Integrated web server
  - Integrated REST client connector to hawkBit[42]
  - Remote server download
- Software delivered as images, gzipped tarball, etc.
- Allow custom handlers for installing FPGA firmware, microcontroller firmware via custom protocols.
- Power-Off safe
- Hardware / Software compatibility.

4.3 Edge Level Software

4.3.1 Edge Service Orchestration

4.3.1.1 Local (manual) Edge Service Orchestration

A virtual machine (VM) is an emulation of a computer system. The operating systems (OS) and their applications share hardware resources from a single host server, or from a pool of host servers. Each VM requires its own underlying OS, and the hardware is virtualized. VM can take up a lot of system resources. With containers, instead of virtualizing the underlying computer like a virtual machine (VM), just the OS is virtualized. Containers sit on top of a physical server and its host OS. Each container shares the host OS kernel and, usually, the binaries and libraries, too. Shared components are read-only. Many containers can turn on one machine and this solves portability and reproducibility problems. Containers are lightweight component, with a startup time in milliseconds or seconds, required less memory space than VM. However, it can be less secure, as the kernel is shared with the host.

[41] https://sbabic.github.io/swupdate/swupdate.html
[42] https://projects.eclipse.org/projects/iot.hawkbit

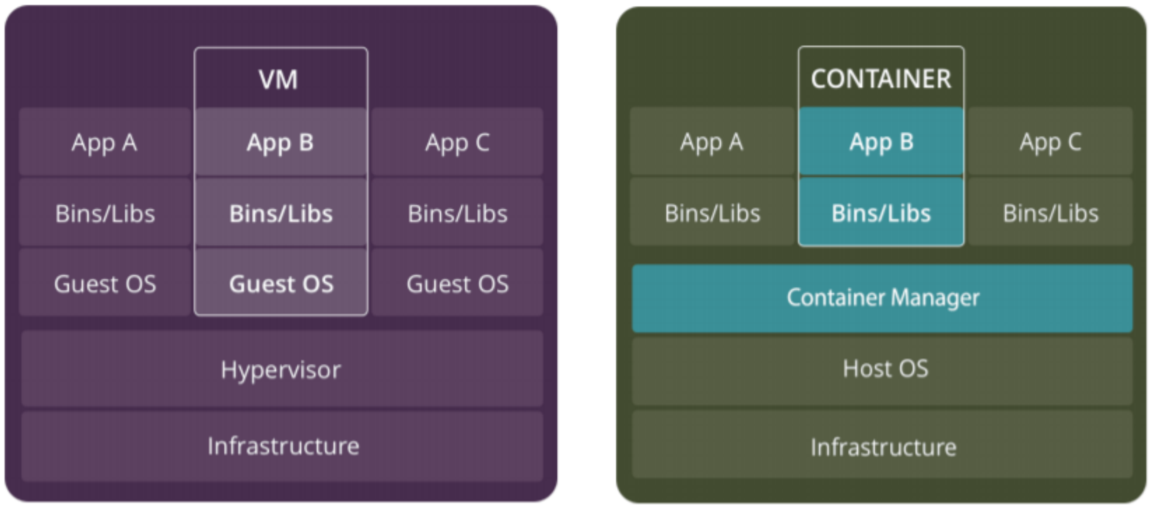


Figure 11: Virtual Machine versus containers (source: [22])

Containers are well adapted in the case when we have many applications running on few servers.

For local management of OCI containers, Docker[43] is a widely used and accepted solution. However, this product has some drawbacks (rather high memory footprint, daemon-centred architecture, usage via root user privileges) which can be blocking for example in connection with resource restricted distributed devices in a security and safety critical environment.

Some alternatives to Docker exist. Singularity[44] is developed by Berkeley specifically for HPC. The user has the same rights inside the container as outside and the tool integrates with the various resource managers (PBS, Slurm, but also Kubernetes). Kata containers[45] is an open source community working to build a secure container runtime with lightweight virtual machines that feel and perform like containers, but provide stronger workload isolation using hardware virtualization technology as a second layer of defence.

Podman[46] is another newly developed solution that can be used instead of docker. Podman is a daemonless, open source, Linux native tool with the goal to make it easy to find, run, build, share and deploy applications using OCI Containers and Container Images. Podman provides a command line interface (CLI) that is similar to the Docker Container Engine CLI to a great extent.

4.3.1.2 Edge Service Orchestration by Kubernetes managed Cloud level: KubeEdge

Container orchestration is about managing the lifecycles of containers, especially in large, dynamic environments. It controls and automates many tasks:

- Provisioning and deploying of containers
- Redundancy and availability of containers
- Scaling up or removing containers to spread application load evenly across host infrastructure
- Movement of containers from one host to another if there is a shortage of resources in a host, or if a host becomes unavailable
- Allocation of resources between containers
- External exposure of services running in a container with the outside world

[43] https://www.docker.com/
[44] https://sylabs.io/docs/
[45] https://katacontainers.io/
[46] https://podman.io/

- Load balancing of service discovery between containers
- Health monitoring of containers and hosts
- Configuration of an application in relation to the containers running it

Kubernetes (K8s[47]) is an open-source system for automating deployment, scaling, and management of containerized applications (OCI containers) in large cluster systems (backends). Thereby the worker nodes on which the applications are running must fulfil one central requirement for the system to work correctly, which is not always fulfilled by edge device: Permanent, stable connectivity to the backend services.

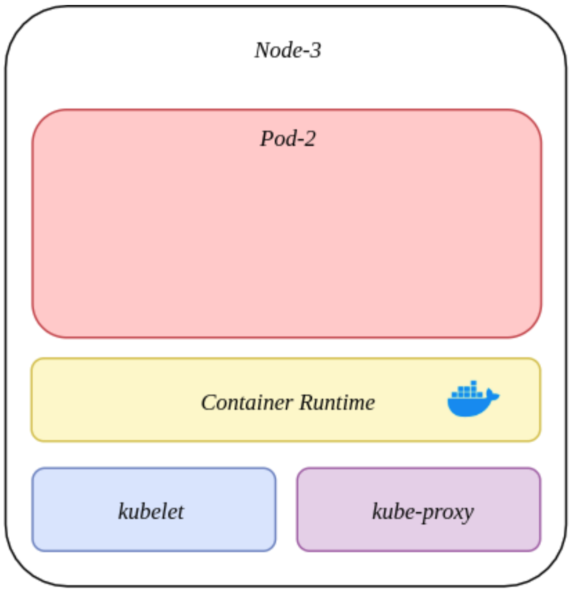


Figure 12: Kubernetes overview (source: [23])

A node is either a virtual or physical machine within the cluster, capable of running container. Nodes run the container runtime that is required (e.g. Docker — although *containerd* is the default), as well as the services *kubelet* (an agent running on every worker node that manages containers which run in pods) and *kube-proxy* (highly flexible proxy that forwards many different kinds of requests and is responsible for handling all interactions between nodes and Kubernetes). The pod is the smallest unit, and it is a running process on the cluster. encapsulates one or several containers, storage resources, a unique network IP and various options. Volume lives inside a pod and allows to share volume between containers.

The control plane or Kubernetes Master is a series of services that run on master nodes. These services control how the k8s software interacts with the cluster. The services primarily consist of:

- **kube-apiserver:** A component responsible for exposing the k8s cluster (frontend).
- **kube-scheduler:** Selects a node for the newly created pods.
- **Etcd:** A distributed key-value store used for configuration management.
- **kube-controller-manager**: A component responsible for managing the lifecycle of pods. *kube-controller-manager* retrieves desired and current cluster state from *etcd* through *kube-apiserver* and instantiates or removes the required resources as necessary.
- **Cloud-controller-manager**: If needed it can interact with cloud providers.

[47] https://kubernetes.io/

KubeEdge is an open source system that aims to extend the containerized application orchestration capabilities of K8s to edge devices. “It is built upon Kubernetes and provides fundamental infrastructure support for network, app. deployment and metadata synchronization between cloud and edge.” [24]. KubeEdge provides fundamental infrastructure support for network, app deployment and metadata synchronization between the backend/cloud and edge devices.

Users can remotely orchestrate apps, manage devices and monitor app and device status on edge nodes from the Kubernetes master. The control plane resides in cloud, though scalable and extendable. At the same time, the edge can work in offline mode. Also, it is lightweight and containerized, and can support heterogeneous hardware at the edge. With the optimization in edge resource utilization, KubeEdge positions to save significant setup and operation cost for edge solutions.

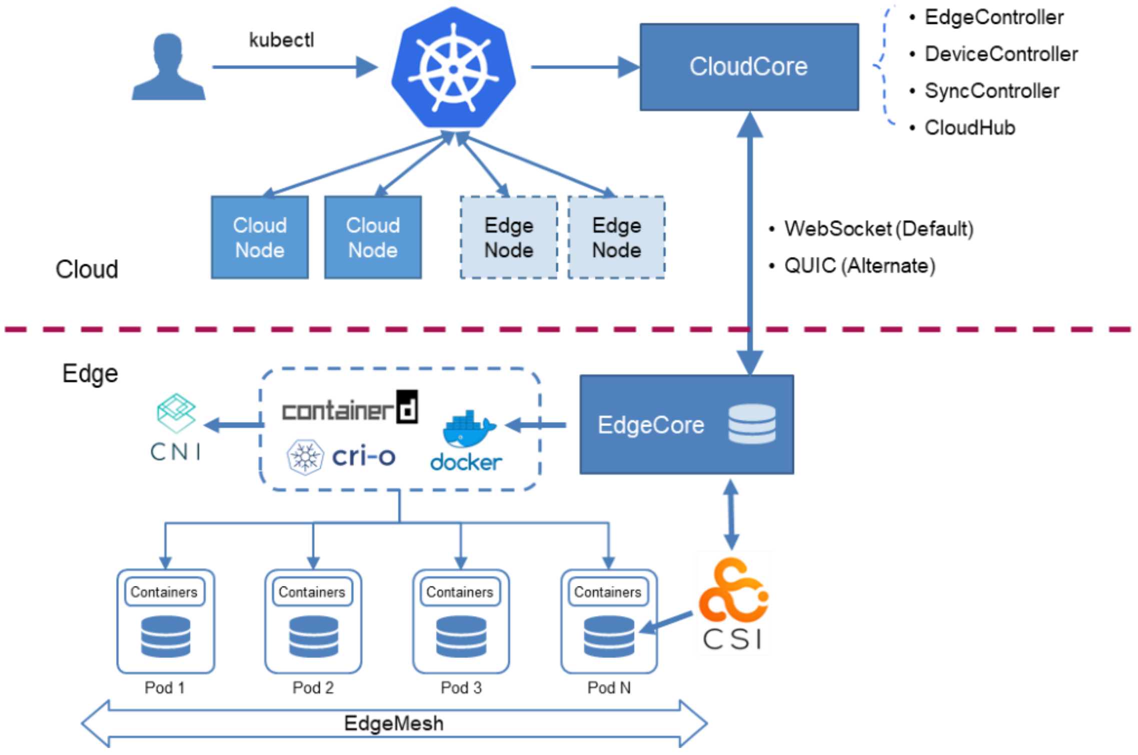


Figure 13: KubeEdge Architecture

The characteristics of the KubeEdge solution are:

- Seamless Cloud-Edge Communication for both metadata and data
- Autonomous operation of edge nodes even during disconnection from cloud. The component that manages the access on the edge node (EdgeCore, see description of components below) uses a local database to store container information, to be able to spin up local containers after disconnection, process errors or reboot.
- KubeEdge can work in constrained resource situations (low memory, low bandwidth, low compute) - 100MB binary size and only 14MB running memory footprint (EdgeCore & CRI-O, see below)
- Support for the ARM processor architecture (64 and 32 bit)

- Reduced network bandwidth requirements and consumption between edge nodes and the cloud services. This increases responsiveness, decreases costs, and protects customers' data privacy.
- Users can orchestrate apps, manage devices and monitor app and device status on edge nodes just like in a traditional Kubernetes cluster in the cloud
- Easy to get and deploy existing complicated machine learning, image recognition, event processing and other high-level applications to the edge.

The cloud level components of the KubeEdge solution that are needed to integrate the edge devices into the Kubernetes backend are shown in Figure 14. The *EdgeController* provides a shadow management for nodes, pods, configmap, etc. at the edge. The *DeviceController* provides IoT Edge device modelling and shadow management for devices at the edge. The *Sync Controller* provides reconcilement for detected inconsistencies. The *KubeEdge CSI Driver* implements a hook storage provisioning to edge devices. The Admission Webhook provides extended API validation and best practice enforcement.

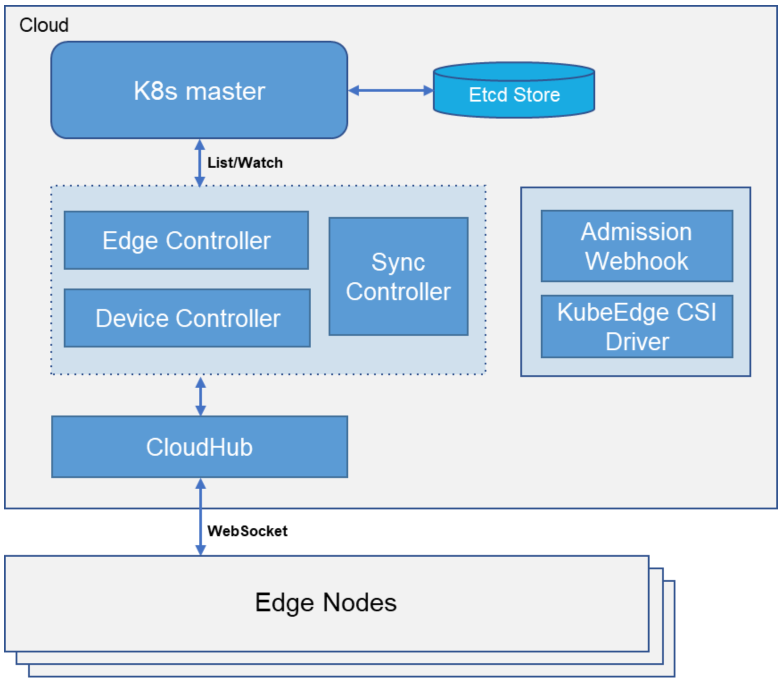


Figure 14: KubeEdge Cloud Level Components

KubeEdge uses the Container Runtime Interface (CRI), a plugin interface which in a traditional Kubernetes node setup enables the *kubelet* to use a wide variety of container runtimes, without the need to recompile. CRI consists of a protocol buffers and gRPC API, and libraries, with additional specifications and tools under active development. CRI has been released in Kubernetes 1.5. Therefore, it is very easy to switch between Container Runtimes depending on the edge node capabilities. The integration of CRI was triggered in the community due to resource limitations using Docker as runtime.

In the context of resource restricted devices, we have selected CRI-O as a lightweight container runtime alternative compared to Docker. CRI-O is optimized for Kubernetes and allows to pull from *any* compliant registry[48] _ and run *any* OCI-compliant container[49]– basically meaning, it is possible to deploy any Docker image using CRI-O.

The Linux Kernel provides tools to be able to run containers natively. For the given edge device not all Kernel container runtime dependencies have been activated. The Moby Project - a collaborative project for the container ecosystem to assemble container-based systems – published a script to check the Linux Kernel for container runtime dependencies[50].

Based on the script and the necessary networking configurations of CRI-O the Linux Kernel could be modified to run OCI-compliant containers on the edge device.

CRI-O does not provide release builds for ARM32v7. In version 1.18.1 we could contribute to the public repository and a bug could be fixed to be able to build and run CRI-O on ARM32v7 without the support of kernel option CONFIG_CGROUP_HUGETLB[51].

In case the KubeEdge environment is running in a separate container using *runc, e.g. for security or safety reasons*, the containers started by KubeEdge and CRI-O must be deployed as rootless containers. CRI-O supports a rootless mode, that can be activated using an environment variable[52] **.**

CRI-O comes with a debugging CLI-tool for deployed containers called `crictl`. With the help of this tool the user can debug Kubernetes deployed containers.

Kubenernetes provides the mount namespace propagation feature: “This feature allows a container to mount a volume as `rslave` so that host mounts can be seen inside the container, or as `rshared` so that any mounts from inside the container are reflected in the host's mount namespace” [25]. To make this default work in all Linux environments the entire mount tree should be marked as shareable. Linux distributions that use systemd already have the root directory mounted as rshared. In Linux environments without systemd it is recommend to run the following command during boot before docker is started:

- `mount –make-rshared /`

Moreover, the filesystem type of the edge device has been changed from VFS to Overlay:

- VFS does not support copy-on-write. To create a new layer, a “deep copy” is done of the previous layer. This leads to lower performance and more space used on disk than other storage drivers, like Overlay.
- Moreover the “deep copy” used by VFS increases container creation time significantly, especially on edge devices.

KubeEdge has not yet implemented all Kubernetes functionalities, for example:

- Metrics are not yet supported for the edge device.
- Due to the registration of an external edge device as Kubernetes node, the integration into existing managed Kubernetes clusters of well-known cloud providers is not fully supported.

[48] https://mcas-proxyweb.us.cas.ms/certificate-checker?login=false&originalUrl=https%3A%2F%2Fcri-o.io.us.cas.ms%2F%23container-images
[49] https://mcas-proxyweb.us.cas.ms/certificate-checker?login=false&originalUrl=https%3A%2F%2Fcri-o.io.us.cas.ms%2F%23oci-compatible-runtimes
[50] https://github.com/moby/moby/blob/master/contrib/check-config.sh
[51] Fix a bug where CRI-O could not start a container if CONFIG_CGROUP_HUGETLB was not set in the kernel (#3721, @haircommander)
[52] `export _CRIO_ROOTLESS=1`

4.3.2 Side-by-side Orchestration of Containerized and Virtualized Workloads at the Edge: KubeVirt

4.3.2.1 Motivation

There is an increasing trend towards more powerful edge installations consisting of servers or even small clusters often referred to as edge appliances or hyper-converged infrastructure. Such systems allow to offload compute-intensive applications from field devices to the edge or to consolidate the functionality of previously separate devices on a common infrastructure.

The notion of hyper-converged edge computing (HCEC) is derived from hyper-converged infrastructures known from cloud computing, which combine compute, storage, and networking capabilities on commercial off-the-shelf (COTS) hardware operated under the roof of a unified management system. HCEC extends classical edge computing by allowing customers to run a variety of workloads on small to medium-sized clusters located near to the field. Key characteristics of HCEC solutions are:

- Low Latency: The geographical vicinity allows to run low latency applications potentially even satisfying real-time requirements.
- Scalability: Users can flexibly deploy their workloads on the cluster and increase (decrease) its capacity by adding (removing) servers on demand.
- Redundancy: There is no single point of failure as the management system handles failover scenarios.
- Efficiency: Hardware utilization is optimized through central resource pooling and load balancing, thus reducing capital expenditures.

HCEC allows to consolidate various kinds of applications currently running on different devices on a single infrastructure. However, in order to satisfy the low and defined latency requirements of some of the applications, the hyper-converged infrastructure may require real-time capable software stacks (hypervisors, operating systems, protocol stacks, etc.) as well as appropriate communication mechanisms such as Time Sensitive Networks (TSN). Figure 15 exemplifies the consolidation of multiple devices using HCEC, where a server runs real-time applications, non-real-time applications, containerized apps, and virtual machines (such as those running Windows or FreeRTOS) on top of a real-time Linux OS enhanced with virtualization support (KVM).

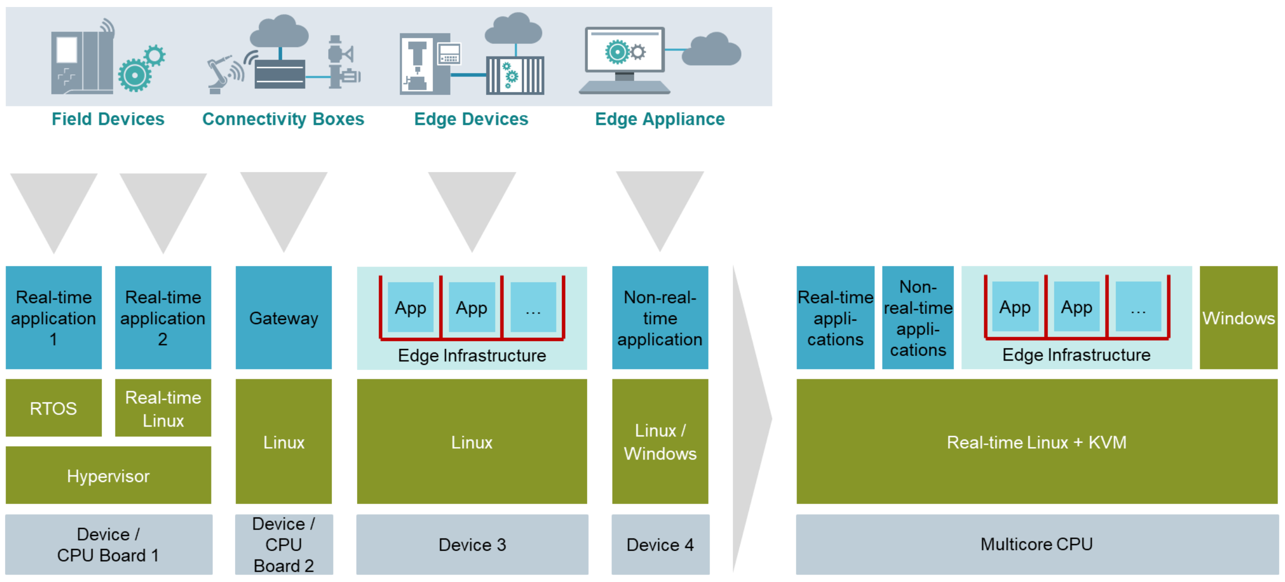


Figure 15: Hardware consolidation in hyper-converged edge computing

Traditionally, cloud providers focused on the orchestration of virtual machines (VMs). Amazon, for example, offers the Elastic Compute Cloud (EC2), where VM images can be imported, exported, and started from a pool of readily available machine images. They can be easily scaled to accommodate changes in load and deployed redundantly in order to satisfy availability constraints. Though response times are of course relevant, real-time requirements are typically not a concern. For edge applications with real-time requirements, local hosting of the compute infrastructure is a prerequisite. As an orchestration system, OpenStack could then be used to deploy VMs, ensure scaling, manage availability and in addition guarantee latencies. However, OpenStack does not support the orchestration of modern containerized workloads. A separate orchestration framework would have to be deployed on top of the infrastructure provided by OpenStack such as Kubernetes. In that case, however, the compute resources between VMs and the Kubernetes cluster are strictly separated. Nodes not in use by Kubernetes cannot be automatically used for deploying VMs. More importantly, the virtualized and containerized workloads are managed through separate and different orchestration systems increasing system complexity, susceptibility to errors, operations costs, thus impacting productivity, efficiency and hardware utilization. As an alternative, applications may be rearchitected into cloud-native applications and to run them exclusively on a container orchestration system. However, this conversion of legacy applications typically leads to high refactoring costs and a delayed time to market.

To solve these problems, there is a need for orchestration systems supporting virtualized as well as containerized workloads and treating them as far as possible alike. One such system is KubeVirt[53] which is essentially an application running inside a Kubernetes cluster. It extends Kubernetes to support VMs. This way, VMs can be scheduled on the worker nodes of the Kubernetes cluster side-by-side with containers. Just like containers in Kubernetes, VMs can run on isolated cores but don't have to. VMs can be deployed redundantly and scaled automatically e.g. depending on the load. Refactoring of virtualized workloads can then happen as needed and a faster time-to-market can be achieved.

4.3.2.2 Use Case: Factory Automation

As a potential use case, consider factory automation. In factory automation, different levels of automation exist. The lowest level of automation is at field level which contains all the devices with sensors to gather data and actuators for moving things around. These field devices are controlled by programmable logic controllers (PLCs) at the next higher level. These PLCs run control tasks which typically have hard real-time constraints and thus require a real-time operating system. The control devices are supervised in a supervisory control and data acquisition (SCADA) system monitoring and controlling the control devices. Above SCADA, manufacturing execution systems (MES) monitor the whole manufacturing process allowing to plan material orders and product shipments. The top-most level integrates all the components and provides information for enterprise resource planning (ERP). These levels are typically depicted in the form of the so-called automation pyramid.

The devices running all these software systems are typically underutilized and consolidation of those devices leads to considerable savings in terms of operational expenditures. Hyper-converged edge infrastructures can be used for running the automation software, leading to better hardware utilization as well as improved scalability and availability. As software at the lower levels typically have real-time requirements, they are most easily migrated using VMs containing the underlying real-time operating system. As a result, VMs have to coexist with containers hosting cloud-native workloads.

[53] https://kubevirt.io/

4.3.2.3 KubeVirt

4.3.2.3.1 Introduction

KubeVirt is designed such that VMs run inside pods. A pod is the smallest deployable unit in Kubernetes that can be created and managed. It contains one or more containers with shared storage and network resources. Since VMs run inside regular pods, they have access to the pod network and storage. KubeVirt uses Kubernetes' support for custom resource definitions (CRDs) in order to define VMs as separate resources in the cluster. The VirtualMachine manifest defines all the resources needed by a VM such as the number of compute cores, the machine type, the amount of memory, the disks and where the disks are stored. The custom resources defined by KubeVirt can be controlled through kubectl as all other resources in the cluster. In contrast to regular pods, VMs can be switched on and off. When VMs are started, a VirtualMachineInstance is created and scheduled to a worker node satisfying all resource constraints. KubeVirt comes with an additional command line utility called virtctl for facilitating the interaction with VMs. It can be used to start and stop VMs and to access the text or graphical consoles.

4.3.2.3.2 Storage

In contrast to containers, VMs typically require persistent storage. Kubernetes defines the concept of persistent volume claims (PVCs) for providing persistent storage[54]. PVCs specify the amount of storage needed. They need to be backed by persistent volumes (PVs), which can either be dynamically or statically provisioned. Dynamic provisioning requires a storage class to be specified in the PVC. Storage classes are resources in Kubernetes as are pods and services and have to be defined by the cluster administrator. Administrators can define a default storage class and alternate storage classes that the user can request in the PVCs. The storage classes then specify a provisioner that is responsible for provisioning the persistent volume requested in the claim. Provisioners are implemented through volume plugins. Kubernetes supports a long list of provisioners out of the box such as GlusterFS, Cinder and volume plugins for large cloud providers like AWS, Azure and Google[55]. KubeVirt currently requires persistent volumes to be of iSCSI type which provide networked block-level access to storage. Dynamic provisioning of iSCSI volumes currently requires installation of a volume plugin that does not ship with Kubernetes[56]. Provisioning iSCSI volumes by hand works out of the box by specifying an iSCSI persistent volume source in the persistent volume specification of the PVC manifest[57]. The serving of block volumes through iSCSI can, for example, be performed by NAS solutions like FreeNAS[58] or TrueNAS [59] running outside of the cluster or by installing containerized block storage solutions like OpenEBS[60] or Longhorn[61] into the Kubernetes cluster.

Importing disk images into PVCs is supported via the Containerized Data Importer (CDI) project[62], which can also be installed into the Kubernetes cluster. It supports the user to import disk images from URLs, from the local disk or to clone existing PVCs. Importing can be set up through the virtctl utility where a target PVC is specified. The CDI controller running in Kubernetes watches for importer specific claims. Upon discovery of such a claim, it starts the import process by creating an importer pod which is responsible for downloading the disk image into the PV bound to the PVC.

[54] https://kubernetes.io/docs/concepts/storage/persistent-volumes/
[55] https://kubernetes.io/docs/concepts/storage/storage-classes/
[56] https://github.com/democratic-csi/democratic-csi
[57] https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.19/#persistentvolumespec-v1-core
[58] https://www.freenas.org/
[59] https://www.truenas.com/
[60] https://openebs.io/
[61] https://longhorn.io/
[62] https://github.com/kubevirt/containerized-data-importer

Besides PVCs also other volumes can be attached to VMs[63]:

- Ephemeral volumes do not persist after the VM stops. A copy-on-write (COW) image is read from a network store and copied locally as disk writes occur. The ephemeral volume is dynamically generated when the VM starts and destroyed after shutdown.
- Container disks also do not persist. A container image is pulled from a container registry. The container image contains a disk image that is made available in the VM. Container disks can be used for replicating VMs.
- Empty disks are sparse qcow2 disks created when the VM starts. It is typically used as temporary disk space e.g. if the root filesystem is read-only.
- Host disks are disk images located directly on the worker nodes of the cluster. If the VM is rescheduled to a different node, the disk image will be a different one.
- Cloud-init disks are disks containing configuration information that are picked up by the VM if the cloud-init services are installed[64]. No Cloud and Config Drive data sources are supported as instance data sources. The user can supply configuration data by various means[65]. Cloud-init supports a large variety of configuration modules allowing the user to configure e.g. the hostname, networking, passwords and ssh access.

4.3.2.3.3 Architecture

The cluster-wide virtualization functionality is provided by the virt-controller operator. Upon creation of a new VirtualMachineInstance (VMI) resource, it is added to the list of all VMI objects managed by the Kubernetes API server. The virt-controller watches this list and creates a virt-launcher pod for each new VMI encountered. The pod will get scheduled to one of the nodes assuming sufficient resources are available. On each node of the cluster, a virt-handler is running as specified in the DaemonSet. The virt-handler signals the creation of the VM to the virt-launcher pod and passes on the resource definition. The *libvirt* daemon running inside the virt-launcher pod is used to start the VM. Afterwards, the virt-handler monitors and controls the libvirt domain to keep it in sync with the VMI resource. Details are given in[66] [67] [68]. Figure 16 sketches the node-level architecture. The operating system on the worker node can be restricted to run on selected cores using the *isolcpus* boot parameter. The CPUManager feature of Kubernetes can be used to run real-time workloads on isolated physical cores[69]. It manages a configurable pool of cores that is shared between workloads not requesting isolation. The cores used by the operating system can be excluded from this shared pool. When real-time workloads are scheduled, these workloads can request exclusive cores. These cores will be removed from the shared pool and the core affinity of all containers managed by Kubernetes will be adjusted to no longer use these exclusive cores. The cpuset of the new container is set in contrast to use exactly the exclusively reserved cores. KubeVirt in addition supports the explicit isolation of the *qemu* emulator thread on a separate core[70] .

[63] https://kubevirt.io/user-guide/#/creation/disks-and-volumes
[64] https://cloudinit.readthedocs.io/
[65] https://kubevirt.io/api-reference/master/definitions.html#_v1_volume
[66] https://kubernetes.io/blog/2018/05/22/getting-to-know-kubevirt/
[67] https://kubevirt.io/2020/KubeVirt-Architecture-Fundamentals.html
[68] https://github.com/kubevirt/kubevirt/blob/master/docs/components.md
[69] https://kubernetes.io/docs/tasks/administer-cluster/cpu-management-policies/
[70] https://kubevirt.io/user-guide/#/creation/dedicated-cpu

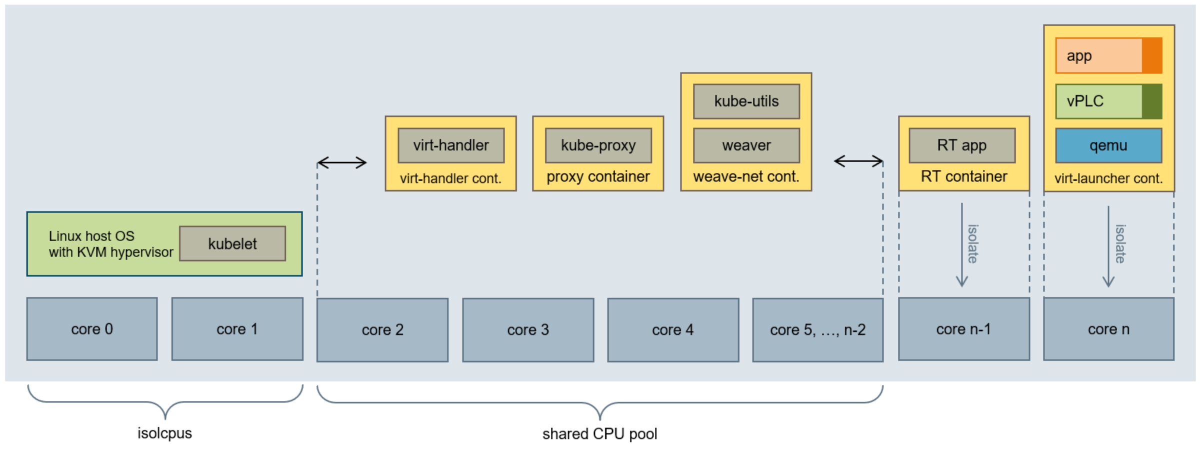


Figure 16: Node-level architecture with isolation of real-time workloads

4.3.3 Edge Architecture for Energy Automation

A typical substation in energy automation nowadays sends its data to a control center (SCADA). For that purpose, data is received via various protocols from the devices (field level). Computing tasks are being executed on local PCs, mostly dedicated to specific machines or automation components. Updating and enhancing the system is often time consuming as each individual PC needs to be updated via USB or DVD or maybe even network. Handling data in a secure manner and dealing with lots of data is the main reason for the current setup of the substations.

4.3.3.1 Overview of SICAM GridEdge

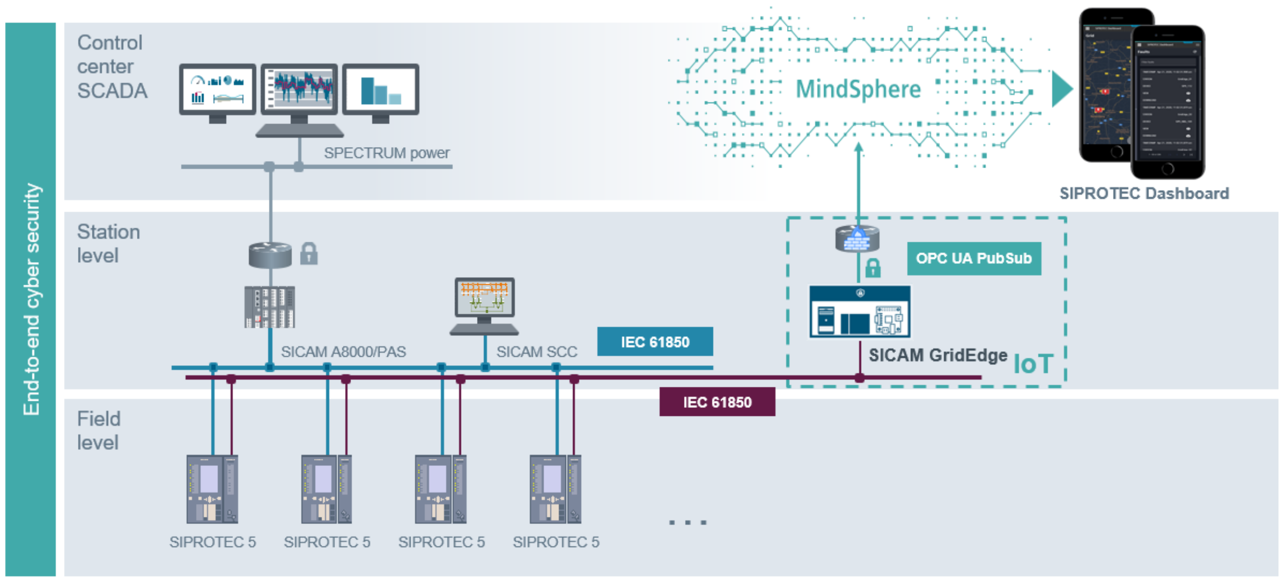


Figure 17: Overview SICAM GridEdge[71]

SICAM GridEdge is a software system based on well proven container technologies. The system connects using IEC61850 (TCP/IP based protocol) to all capable devices (e.g. SIEMENS SIPROTEC5[72] protection devices) in the station network, analyses the data online and publishes the result to an external cloud service in a secured manner. The purpose of SICAM GridEdge is not to replace an existing substation automation with its

[71] https://support.industry.siemens.com/cs/document/109780703/sicam-gridedge-engineering-guide?dti=0&lc=en-PL
[72] https://new.siemens.com/global/en/products/energy/energy-automation-and-smart-grid/protection-relays-and-control/siprotec-5.html

protection features but to enrich the substation by new technologies like IoT and edge/cloud computing. SICAM GridEdge ensures that there is no impact on system-critical features in the substation.

4.3.3.2 Features of SICAM GridEdge

Container technology

Since SICAM GridEdge is based on docker container technology, the system can easily be enhanced and upated.

Data filtering

By providing a pre-defined Data Profile which only collects and forwards relevant data to the cloud system, SICAM GridEdge is not only able to minimize costs due to data transmission and cloud costs, but also manages to decrease the network traffic at station level itself.

Support of IEC61850

The usage of the IEC61850[73] communication in substation automation is widely spread in Europe. A main benefit of this protocol is its openness and self-description of the data. Since many device manufacturers support this standardized protocol, SICAM GridEdge can connect to all IEC61850 capable devices and is not limited to SIEMENS devices.

Furthermore, using the reporting feature of IEC61850, it is no longer needed to poll cyclically for new measurement data which decreases dramatically network load.

Zero configuration approach

Whenever firmware or configuration changes have to be done to protection relays at customer side, it is required to ensure a reliable protection state of the substation by executing a full-blown delivery test. In order to minimize costs, SICAM GridEdge consequently avoids changes on the device side and furthermore provides a user-friendly configuration interface by which a customer can prepare his substation for cloud/edge computing in less than 30 minutes.

Security

As displayed in Figure 17: Overview SICAM GridEdge, SICAM GridEdge has two network connections which are strictly separated from each other. It is not possible to do any controlling actions from outside the station LAN, only measured data is published encrypted to a cloud service using OPC UA PubSub.

Complete Certificate Management

All data transferred to a cloud service is encrypted using TLS/SSL mechanisms. In order to separate data consequently between devices, SICAM GridEdge issues for each device an own certificate. For that purpose, it is only needed to upload a (sub-) certificate authority which later on signs all issued certificates.

[73] https://en.wikipedia.org/wiki/IEC_61850

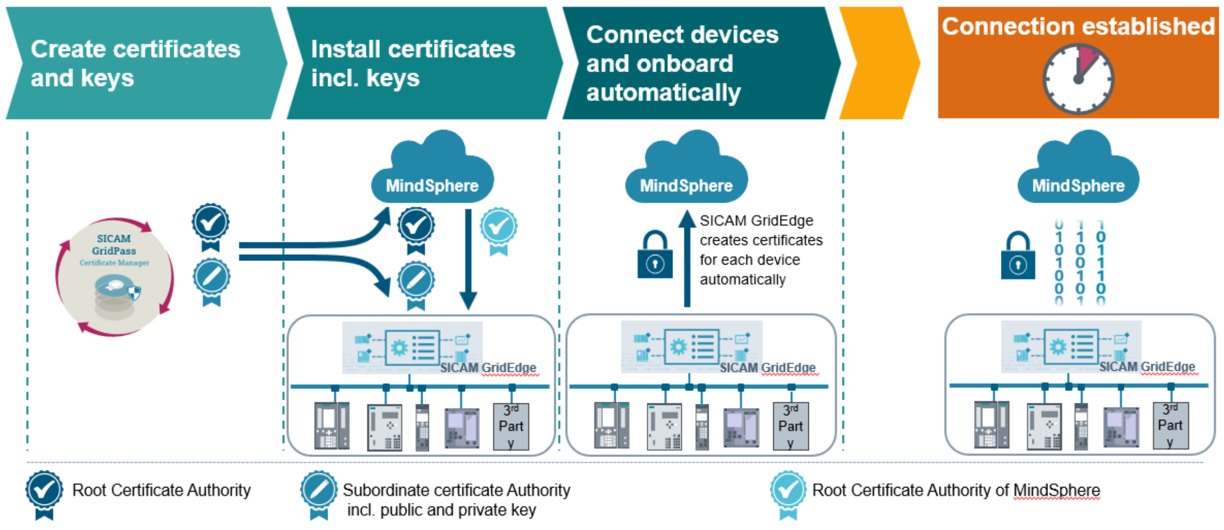

Create certificates and keys
Install certificates incl. keys
Connect devices and onboard automatically
Connection established
SICAM GridPass Certificate Manager
MindSphere
SICAM GridEdge creates certificates for each device automatically
SICAM GridEdge
3rd Party
Root Certificate Authority
Subordinate certificate Authority incl. public and private key
Root Certificate Authority of MindSphere

Figure 18: Certificate Management of SICAM GridEdge[74]

Cloud agnostic

By using the standardized OPC UA PubSub protocol and the related OPC UA 61850 companion, it is possible to easily describe each data point with all essential attributes like device of origin, physical location, topology position, time stamp, measurement type, measurement value etc. Only when the semantics is well-defined and openly accessible, it becomes possible to achieve cross-vendor interoperability and hence a wider adoption. To this end, we picked the evolved version of the well-established OPC UA standard named OPC UA PubSub (MQTT), which as the name suggests wraps the classic OPC UA payload inside an MQTT container. Furthermore, by also adopting and actively contributing to the related OPC UA 61850 companion specification, we get both – a mature data semantic model together with a scalable publish-subscribe architecture.

4.3.4 IoT Level Service Control

As explained in section 3.1, SWUpdate [75], which is a Linux Update agent with the goal to provide an efficient and safe way to update an embedded system, is one way to manage the update of the base system of IoT level devices. In order to be able to manage the updates on the IoT Level without the need to directly connect to the IoT Level device, the Edge Device can provide proxy functionality for these updates. In order to increase security, these proxies could also be started only temporarily for an update process.

4.3.5 Local messaging

Event streaming is the practice of capturing data in real-time from event sources like databases, sensors, mobile devices, cloud services, and software applications in the form of streams of events; storing these event streams durably for later retrieval; manipulating, processing, and reacting to the event streams in real-time as well as retrospectively; and routing the event streams to different destination technologies as needed.

As shown in Figure 9, event streaming through messaging is required to interface the Edge Data Processing component, with its many services, both to the IoT layer and to the Cloud layer. Thus, it is important to choose a messaging technology that achieves high throughput, as events can come in an asynchronous manner from several IoT devices with a high frequency (up to tens of KHz). These events also need to be

[74] https://support.industry.siemens.com/cs/document/109780703/sicam-gridedge-engineering-guide?dti=0&lc=en-PL
[75] https://github.com/sbabic/swupdate

stored. Furthermore, the event processing library will need to output the results of each service of the Data Processing component (ML models, Data Preprocessors) to the cloud-facing interface. In the case of Data Processing, the messages carry measurements that need to be processing in a sequential manner (as time series). This needs to be considered, as imposes latency considerations on the message-passing library.

In order to choose the message processing technology, this section reviews recent well-known open source libraries.

4.3.5.1 Kafka

Apache Kafka[76] combines three key capabilities: (1) To **publish** (write) and **subscribe to** (read) streams of events, including continuous import/export of the data from other systems, (2) to **store** streams of events durably and reliably and (3) to **process** streams of events as they occur or retrospectively.

All this functionality is provided in a distributed, highly scalable, elastic, fault-tolerant, and secure manner. Kafka can be deployed on bare-metal hardware, virtual machines, and containers. Unlike many integration brokers that assume consumers are mostly online, Kafka can successfully persist a lot of data and supports "replay" scenarios.

Kafka allows great flexibility. However, bandwidth overhead is higher than some alternatives that are specifically adapted to the IoT setting.

4.3.5.2 Nats.io

NATS[77] is a lightweight, open-source cloud-based messaging system written in Go language that is open-source. It supports publisher-subscriber, request-reply and messaging queue models.

NATS enables the exchange of data that is segmented into messages among computer applications and services. These messages are addressed by subjects and do not depend on network location. This provides an abstraction layer between the application or service and the underlying physical network. Data is encoded and framed as a message and sent by a publisher. The message is received, decoded, and processed by one or more subscribers.

4.3.5.3 RabbitMQ

RabbitMQ is a message broker system. It is open-sourced and incorporates Advanced Message Queuing Protocol (AMQP). It enables seamless asynchronous message-based communications between applications. The messages transported are loosely coupled, i.e. the sender and the receiver systems need not be up and running at the same time. RabbitMQ persists messages to disk when the receiver is unavailable.

4.3.5.4 MQTT

MQTT[78] is a Client Server publish/subscribe messaging transport protocol. It is light weight, open, simple, and designed so as to be easy to implement. These characteristics make it useful for many situations, including constrained environments such as for communication in Machine to Machine (M2M) and Internet of Things (IoT) contexts where a small code footprint is required and/or network bandwidth is at a premium.

As MQTT is a lower level protocol and not a fully-fledged message processing library, it is possible to integrate MQTT with Kafka. Through Kafka Connect, such as built-in fault tolerance, load balancing, Converters, and Simple Message Transformations (SMT) for routing/filtering/etc., scaling different connectors in one Connect

[76] https://kafka.apache.org/
[77] https://nats.io/
[78] https://mqtt.org/

worker instance and other Kafka Connect related benefits. MQTT and Apache Kafka are complementary and can provide a solution to span the IoT-Edge-Cloud continuum.

MQTT can also be used as protocol for RabbitMQ.

4.3.5.5 Comparison

Depending on the use-case for which the IoTwins architecture is implemented there are different requirements concerning the amount and frequency of data to be transmitted and the tolerable latency for both local and inter-processing-level communication. Thus, one cannot clearly recommend one of the described communication solutions for all use-cases but has to take the features of these solutions into account. A comprehensive comparison of the features of Kafka, RabbitMQ and NATS is detailed in [26]. Benchmarks carried out by [27] show the performance differences between the three libraries in realistic settings with small and large messages.

In the case of small message (up to 16KB in size), Kafka shows good performance with sub millisecond latency for up to 99.99% of the messages. Kafka can handle large messages (>1MB) well, with latencies for a round-trip between 0.5ms and 1.5ms in a 100 requests/second test. For both small and large messages, one performance problem can be seen for a small number of messages (<0.01%) which show a very high latency (10x to 100x more) [27].

NATS shows performance that is more reliable over a range of small to medium message sizes. Thus, the maximum latency encountered for message round-trips is not very different to the average latency obtained. Thus, communication is more reliable, with fewer slowdowns in cases of congestion, and is more adapted for streaming (in sequence) base processing.

RabbitMQ shows similar performance to Kafka, with a large difference in latency for a small number of messages with respect to the average latency. In the case of congestion, this can mean that in-order processing of events can be significantly slowed down. However, RabbitMQ can handle very large message sizes and can use the MQTT protocol to limit bandwidth overhead.

Table 2: Comparison of most popular messaging systems from [26]

Features	Apache Kafka	RabbitMQ	NATS Streaming
Language developed in	Scala	Erlang	Go
Started In	2011	2007	2015
Messaging models supported	pub/sub Message queue	pub/sub Message	pub/sub Message queue request-reply
Brokered/Brokerless	Brokered	Brokered	Brokered
Throughput	High	Medium to High	High
Latency	Low	Low to Medium	High
Protocols supported	Binary over TCP	AMQP, STOMP, MQTT	Google Protocol Buffer

Features	Apache Kafka	RabbitMQ	NATS Streaming
Message size	1 MB max	2 GiB	1 MB max
Message Delivery	At most once, Exactly once, At least once delivery	At most once, At least once delivery	At most once, At least once delivery
Languages supported	About 17 languages	About 30 languages	Officially about 12 languages
Message Ordering	Yes	Yes	Yes
Message Storage	Disk	In-memory/disk	In-memory/disk
Distributed Units	Topics	Queues	Channels
Used By	LinkedIn, Netflix, Facebook, Twitter, Chase Bank	Mozilla, AT & T, Reddit	Baidu, Ericcson, HTC, VMware, Siemens

4.3.6 Device Information Service

A Thing Description describes a virtual or physical device (Thing). It can be considered to be the entry point for a Thing. It describes all available actions, events and properties of a Thing. As well as all available security mechanisms to access them. The Thing Description in highly flexible in order to guarantee interoperability. Additionally, to the standard functionality, it describes, with TD Context Extension, a mechanism to extend the functionality. A Thing Description also contains meta data about a Thing, such as titles and descriptions for the actions, events and properties.

4.4 Cloud Level Software

4.4.1 Bare Metal Application Runtime

In general, an HPC application is designed to yield maximum performance by optimally exploiting the underlying hardware using: parallel programming paradigms such as MPI (or even hybrid MPI/OpenMP) [28], GPU programming [29] and leveraging high-throughput interconnects such as InfiniBand79.

HPC's classical workload is submitted in batch mode directly on bare metal hardware: a user prepares a job that includes a request for resources and commands to execute an application, then submits it through a scheduler, such as Slurm [30]. When the resources requested by the user are available, the job is pushed to compute nodes to be executed. The scheduler is extremely important since its task is to allocate and manage the whole system's resources, improving the performance and filling the capacity of an HPC cluster; a modern scheduler can manage thousands of compute nodes.

Compared to the use of Cloud, HPC can have some advantages as long as there is a flexibility loss. However, for large data moles or intensive computational algorithms it is necessary and enables:

79 https://en.wikipedia.org/wiki/InfiniBand

- The data can be read and written in parallel through the parallel file system, such as GPFS[80] or LUSTRE[81], from the compute nodes
- The software is optimized on the architecture and is ran without a virtual layer. All resources are dedicated to the application.
- The AI training or data analysis can leverage more than one compute node because it can synchronize data and results through the high-performance network [31].

4.4.2 Mesos-Marathon and Openstack

Apache Mesos[82] is an open-source project to manage computer clusters. Marathon[83] is a production-grade container orchestration platform for Mesosphere's Datacenter Operating System (DC/OS) and Apache Mesos. Marathon is a framework (or meta framework) that can launch applications and other frameworks. and can also serve as a container orchestration platform which can provide scaling and self-healing for containerized workloads.

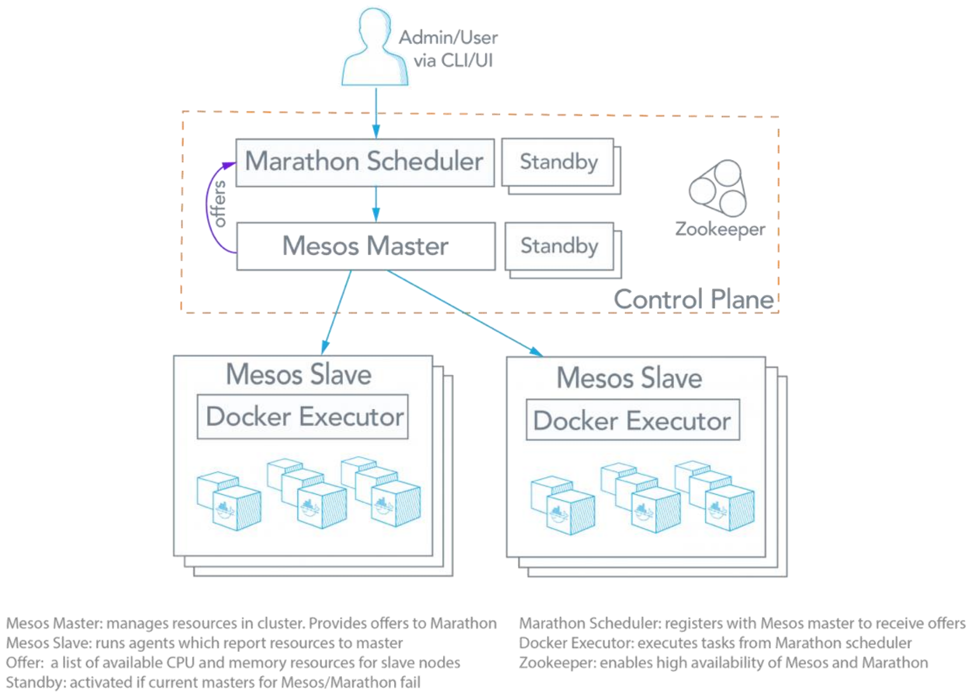

Admin/User via CLI/UI
Marathon Scheduler
Standby
offers
Mesos Master
Standby
Zookeeper
Control Plane
Mesos Slave
Docker Executor
Mesos Slave
Docker Executor
Mesos Master: manages resources in cluster. Provides offers to Marathon
Mesos Slave: runs agents which report resources to master
Offer: a list of available CPU and memory resources for slave nodes
Standby: activated if current masters for Mesos/Marathon fail
Marathon Scheduler: registers with Mesos master to receive offers
Docker Executor: executes tasks from Marathon scheduler
Zookeeper: enables high availability of Mesos and Marathon

Figure 19 Marathon and Mesos Architecture[84]

Mesos CLI or UI can be used. Docker containers can be launched using JSON definitions that specify the repository, resources, number of instances, and command to execute. Scaling-up can be done by using the Marathon UI, and the Marathon scheduler will distribute these containers on slave nodes based on specified criteria. Autoscaling is supported. Multi-tiered applications can be deployed using application groups.

Containers can be scheduled without constraints on node placement, or each container on a unique node (the number of slave nodes should be at least equal to the number of containers).

[80] https://www.ibm.com/products/scale-out-file-and-object-storage
[81] http://lustre.org/
[82] http://mesos.apache.org/documentation/latest/
[83] https://mesosphere.github.io/marathon/
[84] https://medium.com/@axbaretto/being-recently-asked-to-do-a-comparison-of-kubernetes-with-the-mesos-marathon-container-c5a8a58c44e

High availability for Mesos and Marathon is supported using Zookeeper[85]. Zookeeper provides election of Mesos and Marathon leaders and maintains cluster state.

Host ports can be mapped to multiple container ports, serving as a front-end for other applications or end users.

Marathon continuously monitors the number of instances of a Docker container that are running. If one of the containers fail, Marathon reschedules it on other slave nodes. Auto-scaling using resource metrics is available through community-supported components only.

Local persistent volumes (beta) are supported for stateful applications such as MySQL. When needed, tasks can be restarted on the same node using the same volume.

The use of external storage, such as Amazon EBS, is also in beta. At the present time, applications that use external volumes can only be scaled to a single instance because a volume can only attach to a single task at a time.

On the Cloud side, within IoTwins cloud providers, the preferred solution is based on the OpenStack framework, even if for replicability purposes other implementation are acceptable provided that the APIs exposed are compatible with the adopted Orchestration services.

OpenStack is an open source platform that uses pooled virtual resources to build and manage private and public clouds. It mostly is used to deploy an infrastructure-as-a-service (IaaS) cloud service model. The tools that comprise the OpenStack platform, called "projects," handle the core cloud-computing services of compute, networking, storage, identity, and image services. In example the "project" Nova is the OpenStack project that provides a way to provision compute instances (aka virtual servers). Nova supports creating virtual machines, bare metal servers (through the use of ironic), and has limited support for system containers. Nova runs as a set of daemons on top of existing Linux servers to provide that service. Neutron is an OpenStack project to provide “network connectivity as a service” between interface devices (e.g., vNICs) managed by other OpenStack services (e.g., nova). It implements the OpenStack Networking API. It manages all networking facets for the Virtual Networking Infrastructure (VNI) and the access layer aspects of the Physical Networking Infrastructure (PNI) in the OpenStack environment. OpenStack Networking enables projects to create advanced virtual network topologies which may include services such as a firewall, and a virtual private network (VPN). Cinder is the OpenStack Block Storage service for providing volumes to Nova virtual machines, Ironic bare metal hosts, containers and more.

Keystone is an OpenStack service that provides API client authentication, service discovery, and distributed multi-tenant authorization by implementing OpenStack's Identity API. However, can be integrated with other Identity Managements systems including LDAP and OIDC based services, including Indigo-IAM.

Users can interact with the various OpenStack components via a web GUI or via native APIs. However, for several projects AWS compatibles APIs exists, in particular, for managing computing resources, in addition to the native compute API, OpenStack provides an EC2-compatible API. This API allows EC2 legacy workflows built for EC2 to work with OpenStack. Nova in tree EC2-compatible API is deprecated. The ec2-api project is working to implement the EC2 API.

[85] https://zookeeper.apache.org/

Details on all the OpenStack components and projects can be found on[86].

Within IoTwins, each testbed will have access to dedicated tenant on the OpenStack-managed resources at cloud provides allowing isolation between the testbeds and avoiding dependencies that could impact on the replicability of the computing infrastructure for each testbed. When requested by security requirements within the single testbed, multiple tenants can be instantiated.

According to the TBs needs, HPC resources will be provided by the datacenter participating into the project for simulations and to run machine learning applications.

CINECA resources include both Cloud and HPC:

- Marconi100 partition is composed by 980 nodes interconnected by Mellanox IB EDR DragonFly++ 100 Gbit/s network, each one is equipped with 2x16 cores IBM POWER9 AC922 at 2.6 GHz, 4 x NVIDIA Volta V100 GPUs/node and 256 GB of RAM. The file system is mounted via GPFS.
- Cloud HPC is a CPU based system composed by 80 nodes, managed by Openstack, interconnected by Mellanox Ethernet 25 GBs network, each one is equipped with 2 x 18-cores Intel Xeon E5-2697 v4 (Broadwell) at 2.30 GHz and 250 GB of RAM. Complete the infrastructure a dedicated CEPH storage of 200 TB capacity in high availability.

CNAF resources includes about 100 X86_64 CPU cores and 8 NVIDIA V100 GPUs for the computing requirements and 1PB of raw disk space for data storage. Low latency Infiniband interconnect is available among the storage and computing nodes. Access will be granted though Cloud interfaces Openstack APIs or GUI. Data transfer will be possible through standard protocols, I.e. HTTPS, MQTT, S3.

BSC-CNS hosts MareNostrum, the most powerful supercomputer in Spain. At the end of June 2017 begun operating MareNostrum 4, which has a peak performance of 13.7 Petaflops.

Its calculation capacity is distributed in two completely different blocks: a general-purpose block and an emerging technologies block:

1. The general-purpose block has 48 racks with 3,456 nodes. Each node has two Intel Xeon Platinum chips, each with 24 processors, amounting to a total of 165,888 processors and a main memory of 390 Terabytes. Its peak power is 11.15 Petaflops.
2. The second element of MareNostrum 4 is formed of clusters of three different emerging technologies that will be added and updated as they become available. These are technologies currently being developed in the US and Japan to accelerate the arrival of the new generation of pre-exascale supercomputers.

4.4.3 INDIGO PaaS Orchestrator

The INDIGO PaaS Orchestrator[87] is a component of the PaaS layer that allows to instantiate resources on Cloud Management Frameworks (like OpenStack and OpenNebula) and Mesos clusters. It takes the deployment requests, expressed through templates written in TOSCA YAML Simple Profile v1.0, and deploys them on the best cloud site available. In order to do that:

- it gathers SLAs, monitoring info and other data from other platform services
- it asks to the cloud provider ranker for a list of the best cloud sites.

[86] https://www.openstack.org/software/
[87] https://indigo-dc.gitbook.io/indigo-paas-orchestrator/

The Orchestrator depends on various services such as:

- SLAM (SLA Manager): allows to retrieve all the SLAs of the user
- CMDB (Configuration Manager DataBase): contains all the cloud sites information, like the identity endpoint, the OCCI endpoint, etc...
- Zabbix Wrapper (REST wrapper for Zabbix): allows to retrieve monitoring metrics to zabbix through a REST interface
- CPR (Cloud Provider Ranker): it receives all the information retrieved from the aforementioned services and provides the ordered list of the best sites

The language in which the INDIGO PaaS receives end user requests is TOSCA [88]. TOSCA stands for Topology and Orchestration Specification for Cloud Applications. It is an OASIS specification for the interoperable description of applications and infrastructure cloud services, the relationships between parts of these services, and their operational behaviour. TOSCA has been selected as the language for describing applications, due to the wide range of adoption of this standard, and since it can be used as the orchestration language for both OpenNebula (through the IM) and OpenStack (through Heat). The released INDIGO PaaS layer is able to provide automatic distribution of the applications and/or services over a hybrid and heterogeneous set of IaaS infrastructures, on both private and public clouds. The PaaS layer is able to accept a description of a complex set, or cluster, of services/applications by mean of TOSCA templates, and is able to provide the needed brokering features in order to find the best fitting resources. During this process, the PaaS layer is also able to evaluate data distribution, so that the resources requested by the users are chosen by the closeness to the storage services hosting the data requested by those specific applications/services.

5 Conclusion

This deliverable reports on the effort made so far to build an architecture which on the one hand considers the heterogeneity of the different testbeds and at the same time defines standard based interfaces for functionality blocks to allow for further development and integration of 3rd party tools.

Following a proven software development approach, the requirement collection and analysis phase was followed by applying techniques like use case diagrams and creation of a general high-level architecture. Detailing this architecture and applying it to the three levels the platform has to deal with, was the centre of activity and is described in section 3.

The collection of software available to fulfil the functions identified in section 4, allows for a short analysis of gaps so far and where the emphasis on development activities within the project should be.

For the IoT level:

- As stated above the testbeds are very heterogenous and this is true at IoT level especially. Sensors and Actuators cannot be provided by the platform as they are too specific for each use case. Therefore, **Data Collection** at first is covered by each testbed.
- No runtime environments dedicated to the IoT level have been provided so far, thus finding a suitable solution to enable **Computation** is an open task for WP2. Management of IoT nodes on a lower level can be supported by SWUpdate (section 4.2.1).

[88] http://docs.oasis-open.org/tosca/TOSCA/v1.0/TOSCA-v1.0.html

- Also, Gateway functionality and communication enablers used to provide the **Data Transportation** functional block are not covered so far by the tools provided by the project's partners.

For the Edge level:

- Databases and their adaption to the **Data Processing** entities have to be identified together with the service developers of WP3.
- Tools that provide software for edge node infrastructure provisioning as well as **Management of services provided on the Edge level** is described in section 4.3.1.
- **Management of functionality on the IoT level** can, if supported by the IoT device, may be done by SWupdate.
- Tools and services to provide **Communication infrastructure to both IoT and Cloud** are described in section 4.3.5, potentially needed gateway functionality has to be developed during the course of the project.

On Cloud level:

- The cloud level can be considered most mature as solutions like OpenStack or Mesos Marathon, that cover many of the identified functional blocks (**Data Access**, **Resource Management**, Runtime **for Data Processing** Applications), are existing for some time now and are already in place at the cloud providers facilities.
- For **Service Orchestration**, the Indigo PaaS Orchestrator provides a suitable alternative.
- As on the edge level, the Streaming Data Adapters for **Communication with the Edge devices** via different communication protocols have to be developed during the project's life time in collaboration with WP3.

In the following work, it is planned to build a prototype platform which then can be deployed and tested with the project’s use case testbeds. Experiences and discussions resulting from these prototyping will directly influence the next development iteratively. The next version of this report, due in October 2021 will summarize and report on this.

6 References

[1] R. Srikant and R. Agrawal, “Privacy preserving data mining,” in *ACM SIGMOD Conference*, 2000.

[2] Q. Zhang, N. Koudas, D. Srivastava and T. Yu, “Aggregate query answering on anonymized tables,” in *IEEE 23rd International Conference on Data Engineering, April 2007, pp. 116–125.*, 2007.

[3] C. Aggarwal and P. S. Yu, “A survey of randomization methods for privacy-preserving data mining,” *Privacy-Preserving Data Mining - Models and Algorithms,* p. 137–156, 2008.

[4] K. Liu, C. Giannella and H. Kargupta, “A survey of attack techniques on privacy-preserving data perturbation methods,” in *Privacy-Preserving Data Mining*, 2008, pp. 359-381.

[5] C. Dwork, “ Differential Privacy,” Berlin Heidelberg,, 2006.

[6] A. Roth and C. Dwork, “The algorithmic foundations of differential privacy,” in *Foundations and Trends® in Theoretical Computer Science*, 2014.

[7] L. Sweeney, “K-anonimity: A model for protecting privacy,” *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems, vol. 10, no. 05, .,* vol. 10, no. 5, p. 557–570, 2002.

[8] A. Machanavajjhala, D. Kifer, J. Gehrke and M. Venkitasubramaniam, ““l-diversity: Privacy beyond k-anonymity,”,” in *IN ICDE*, 2006.

[9] N. Li, T. Li and S. Venkatasubramanian, “t-closeness: Privacy beyond kanonymity and l-diversity,” in *IEEE 23rd International Conference on Data Engineering, April 2007*, 2007.

[10] V. Torra and J. Domingo-Ferrer, ““Ordinal, continuous and heterogeneous k-anonymity through microaggregation,”,” *Data Mining and Knowledge Discovery,* vol. 11, no. 2, p. 195–212, 2005.

[11] C. Castelluccia and G. Acs, “A Case Study: Privacy Preserving Release of Spatio-temporal Density in Paris,” in *KDD '14: Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2014.

[12] “Pseudonymisation techniques and best practices,” ENISA report, November 2019 .

[13] V. Ciriani, :. S. D. C. d. Vimercati, S. Foresti, S. Jajodia, S. Paraboschi and P. Samarati, “Fragmentation and Encryption to Enforce Privacy in Data Storage,” *ESORICS 2007,* vol. 4734.

[14] G. Memmi, K.Kapusta and H.Qiu, “Data Protection: Combining Fragmentation, Encryption, and Dispersion, an intermediary report,” ITEA2-CAP WP3 Intermediary Report, June 2015.

[15] E. Barker, W. Barker, W. Burr, W. Polk and M. Smid, “Recommendation for key management: Part 1: General. National Institute of Standards and Technology,” 2006.

[16] K. Kapusta and G. Memmi, “Circular AON: A very fast scheme to protect encrypted data against key exposure,” in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, October.

[17] G. Karame, C. Soriente, K. Lichota and S. Capkun, “Securing cloud data under key exposure,” *IEEE Transactions on Cloud Computing.,* 2017.

[18] A. Biryukov and L. Perrin, “State of the Art in Lightweight Symmetric Cryptography,” *IACR Cryptol. ePrint Arch., 2017, 511.,* 2017.

[19] J.-P. Aumasson and A. Vennard, “Cryptography in Industrial Embedded Systems Our experience of needs and constraints,” in *Lightweight Cryptography workshop, NIST*, 2019.

[20] C. Gentry and D. Boneh, “A fully homomorphic encryption scheme,” vol. 20, no. 9, pp. 1-209, 2009.

[21] P. Grubbs, T. Ristenpart and V. Shmatikov, “Why Your Encrypted Database Is Not Secure,” New York, NY, USA, 162–168., 2017.

[22] “Backblaze.com - VM vs Containers,” [Online]. Available: https://www.backblaze.com/blog/vm-vs-containers/).

[23] “Kubernetes Overview,” [Online]. Available: https://medium.com/better-programming/yet-another-kubernetes-k8s-guide-52377a72ce65.

[24] “Kubeedge.io,” [Online]. Available: https://kubeedge.io/en/.

[25] T. K. Authors, "Kubernetes Blog - Release Kubernetes 1.10," The Linux Foundation, 02 March 2018. [Online]. Available: https://kubernetes.io/blog/2018/03/first-beta-version-of-kubernetes-1-10/. [Accessed 01 July 2020].

[26] T.Sharvari and K.Sowmya, A Study on Modern Messaging Systems - Kafka, RabbitMQ, NATS Streaming, arvix.rg, 2019.

[27] T. Treat, “Benchmarking Message Queue Latency,” [Online]. Available: https://bravenewgeek.com/benchmarking-message-queue-latency.

[28] R. Rabenseifner, G. Hager and G. Jost, “Hybrid MPI/OpenMP Parallel Programming on Clusters of Multi-Core SMP Nodes,” *2009 17th Euromicro International Conference on Parallel, Distributed and Network-based Processing,* pp. 427-436.

[29] D. J. Sanders and E. Kandrot, CUDA by Example: An Introduction to General-Purpose GPU Programming, 2011.

[30] A. B. Yoo, M. A. Jette and M. Grondona, “Slurm: Simple linux utility for resource management,” in *Workshop on Job Scheduling Strategies for Parallel Processing*, Heidelberg, 2003.

[31] A. Sergeev and M. D. Balso, Horovod: fast and easy distributed deep learning in TensorFlow, arXiv preprint, 2018.

[32] C. &. B. D. Gentry, “A fully homomorphic encryption scheme,” vol. 20, no. 9, p. 209, 2009.

[33] K. &. M. G. Kapusta, “Circular AON: A very fast scheme to protect encrypted data against key exposure.,” in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, October.