BRAINE

BRAINE - Big data Processing and Artificial Intelligence at the Network Edge

Project Title:	**BRAINE - Big data Processing and Artificial Intelligence at the Network Edge**
Contract No:	876967 – BRAINE
Instrument:	ECSEL Research and Innovation Action
Call:	H2020-ECSEL-2019-2-RIA
Start of project:	1 May 2020
Duration:	36 months

Deliverable No: D4.2

Second report on the status of WP4

Due date of deliverable:	31 March 2022
Actual submission date:	12 April 2022
Version:	1.0

Project funded by the European Community under the H2020 Programme for Research and Innovation.

Project ref. number	876967
Project title	BRAINE - Big data Processing and Artificial Intelligence at the Network Edge

Deliverable title	First report on the status of WP4
Deliverable number	D4.2
Deliverable version	Version 1.0
Previous version(s)	-
Contractual date of delivery	31 March 2022
Actual date of delivery	12 April 2022
Deliverable filename	D4.2 Second report on the status of WP4
Nature of deliverable	Report
Dissemination level	PU
Number of pages	59
Work package	WP4
Task(s)	T4.1, T4.2, T4.3, T4.4

Partner responsible	LUH
Author(s)	Javad Chamanara (LUH), Ahmed Khalid (DELL), Sean Ahearne (DELL), Edgard Marx (ECC), Ilya Vershkov (NVIDIA), Janina Habrunner (IFX), Philippe Nguyen (SIC)
Editor	

Abstract	This technical report, delivers the detailed information about the progresses that have been made in the context of work package 4 (User-oriented utilization of the edge). The report covers architectural and technological designs, decisions, selections, and developments for various aspects of the work package including, distributed data storage, data access and privacy control, workflow definition and execution for managing scalable workloads. It also covers the advancements regarding the workflow definition language and authoring tool, its integration with the service catalogue, and the ontology to describe the cluster concepts. The report provides implementation details about the telemetry and monitoring developments with an emphasis on the monitoring information visualization.
Keywords	Edge computing, resource utilization, distributed file system, privacy and security aware data access, workflow definition, workflow execution, telemetry, monitoring, visualization.

Copyright

Deliverable history

Version	Date	Reason	Revised by
00	24.02.2021	Table of Contents - version 00	Javad Chamanara
01	10.03.2021	Contributions, all partners	Javad Chamanara
1.0	11.04.2022	Executive summary and Final Review	Javad Chamanara, F. Cugini

List of abbreviations and Acronyms

Abbreviation	Meaning
5G	5th Generation
AI	Artificial Intelligence
API	Application Programming Interface
CPU	Central Processing Unit
CU	Centralized Unit
DSP	Digital Signal Processors
DU	Distributed Unit
ECG	ElectroCardioGram
EEG	ElectroEncephaloGram
EMDC	Edge Mobile Data Center
EPC	Evolved Packet Core
ERP	Enterprise Resource Planning
EU	European Union
FPGA	Field Programmable Gate Arrays
GDPR	General Data Protection Regulation
GPU	Graphics Processing Unit
HRC	Human-Robot Collaboration
iDT	intelligent Digital Twin
ICT	Information and Communication Technologies
IP	Internet Protocol
IoMT	Internet of Medical Things
IoT	Internet of Things
IT	Information Technology
KPI	Key Performance Indicator

MES	Manufacturing Execution Systems
MOD	MOtif Discovery
PoC	Proof of Concept
QSD	Qualified Synthetic Data
RAN	Radio Access Network
TBC	To Be Confirmed
TBD	To Be Defined
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TFLOPS	Tera Floating Point Operations Per Second
TSN	Time-Sensitive Networking
UE	User Equipment
URI	Uniform Resource Identifier
URLCC	Ultra-Reliable Low-Latency Communication
USRP	Universal Software Radio Peripheral

Table of Contents

List of Figures

1. Executive summary

This work package aims at developing techniques and tools to enable easy and efficient access to the resources and services provided by the infrastructure to the end-users. This cover providing an easy mechanism for specifying workloads, a secure, scalable, and harmonized data management system that is seamlessly integrated into the infrastructure, and a set of well-tailored data curation and cleansing mechanisms, as well as simplifying the development of systems and applications at the edge of the network.

To achieve its objectives, the work package has been divided into four tasks; 1) secure data management framework for AI at the Edge (started on M03), 2) efficiently handling data at the edge at scale (started on M07), 3) workflow definition language and authoring tool (started on M07), and 4) execution measurement and monitoring (started on M12). The document has correspondingly divided into four sub sections, each detailing the achieving of their respective tasks.

T4.1 provides the details of the status of the implementation of the data lifecycle management and the distributed storage system. It also explains how the file system is integrated with the policy management sub-system and how the distributed file system is accessible as a single unit. A first revision of the active data product (ADP) is also presented.

T4.2 Provides the specification and realization of a data placement framework that is able to place data on nodes based on privacy and security constraints. The partners have done this by extending the APIs of Apache Ozone in a way that Ozone will call the placement algorithm during the file access request pipeline. Also, the task had delivered a motif discovery tool that analyses the time-series to find repeating patterns (motifs) and builds a dictionary of distinctive motifs shipped to the cloud. In order to integrate vRAN with BRAINE, use-case requirements and technical specification have been identified and two approaches have been considered: baseline approach and extended approach. This task, has designed and is implementing an AI application profiling to explore the optimization space and ensure optimal hardware utilization.

T4.3 has delivered a workflow definition language based on OWL-S for modelling atomic, simple, and composite processes. It is integrated with the Kubernetes-compatible open-source workflow execution framework Argo. The authoring frontend allows for defining various work items, e.g., services, and deployments and is able to submit them to the Kubernetes APIS for execution.

In T4.4, P4 based programs for collecting and exporting telemetry records from the switches have been developed and tested. The metrics are successfully submitted to and ingested by the central telemetry database. The flow telemetry agent is able to identify and collect per-flow metrics, which enables higher management modules to decide on rerouting the traffic or transferring the workload to other nodes to maintain SLAs. Monitoring and dashboarding are working at the cluster level. They are Dockerized, Podified, and registered in the cluster-level docker registry. Efforts to make them multi-tenant as well as applying access control on their data have been recently started.

2. Secure data management framework for AI at the Edge

2.1. Background

This task focuses on developing a data storage solution for the BRAINE platform and a framework for managing BRAINE data throughout its lifecycle. Various issues are considered when designing the solution including, access policies, storage policies, data archival, data privacy, data provenance, and data sovereignty.

“D4.1 First report on the status of WP4” provides a detailed background information on the three key elements of this task, as summarized below:

Storage: The distributed data storage system of BRAINE can store data across multiple physical nodes or EMDCs forming a cluster of storage units. Distributed storage has several advantages compared to its centralized counterpart, such as scalability, reliability, performance, cost, fault tolerance, high availability, and disaster recovery. Apache Ozone was selected as the filesystem for the BRAINE distributed data storage solution. It is a relatively new open-source storage system maintained by Apache foundation. Similar to CephFS, it provides object, block, and file storage on one system and supports Native K8S integration. It is highly compatible with HDFS, Amazon S3, and Network file systems.

Data lifecycle and Policy Manager: The concept of data lifecycle management stems from the requirement that the current state of any and all user data contained within the platform needs to tracked to ensure compliance with any user and/or regulatory data policies. The underlying principles can be defined as standard policies, and applied to the data lifecycle monitoring system to ensure that the behavior of both the monitoring system and user data is compliant with all current data regulations. To monitor the lifecycle of data means to maintain a log of all actions and events that transpired with that data, from its point of ingestion into the platform to the point it is archived or deleted. In BRAINE platform the immutable logs are maintained using blockchain and Apache Ranger is used to maintain and apply policies to the data processing components.

Data Security and Privacy: Various security and encryption mechanisms are explored to protect the data and privacy of the platform users. Some of the common security and privacy threats are unauthorized access, data leaks, side-channel attacks, inference attacks, linking attacks, user authentication, access control and compromising data confidentiality or integrity. The storage system is designed to allow incorporation of mechanisms that protect user data against such attacks.

2.2. System Design

2.2.1. Data lifecycle management

The objectives of the data lifecycle manager remain the same as specified in D4.1, which is the tracking, tagging, and immutable logging of data lifecycle events of user data as it is used on the BRAINE platform. In order to achieve this, a number of BRANE platform data management components need to be integrated:

1. The lifecycle management component must take input from the policy manager in order to determine which data should be monitored, what tags exist and should be attached to said data, and detect any possible policy violation such as modification of data from an unauthorized user.
    - Through the use of Apache Ozone for the global filesystem, Apache Ranger can be used as the global data policy manager, as it already has a native plug-in to connect with Ozone.
    - Ranger can act as a simpler server and database to enable users or the platform to upload data policies (with built-in policy auditing), which can be retrieved by other platform components to apply or enforce these policies (e.g. the global filesystem and the lifecycle manager).
2. Input from the global file system is also required in order to determine when a data lifecycle event has occurred for a piece of data. The architecture of apache Ozone allows this to be enabled in many different ways through the use of its Recon server, a global managing and monitoring console for Ozone:
    - Tracking of user data events at the block level can be enabled through the use of the Storage Container Manager database.
    - Tracking of data events at the container level via the Recon server's container database.
    - Tracking of data events at the volume level with the Ozone Manager database.
    - It may be possible to define the level of granularity required in terms of tracking of certain data via the policy manager.
3. Once the metadata on user data is defined and collected from the global file system, it is ready to be ingested to an immutable Ledger in order to ensure verifiable auditability:
    - This will be achieved through ingesting the collected events onto a blockchain.
    - IOTA has been chosen in this instance to provide a lightweight implementation of blockchain technology. Its ability to run as a container and low resource usage make it suitable for both the Edge environment and the BRAINE platform.
    - Modification of the data being ingested, or modification of the data contained within a blockchain transaction may be required in order for the system to operate in an effective manner without excess resource usage.

Figure 2.1 below describes this updated view of the overall integration process for the data lifecycle manager in the context of D4.1, and the overall platform architecture. It should be noted that the blockchain implementation in this instance is private. This provides privacy on behalf of the administrator and users of the platform, but requires more active monitoring in order to ensure effective scalability.

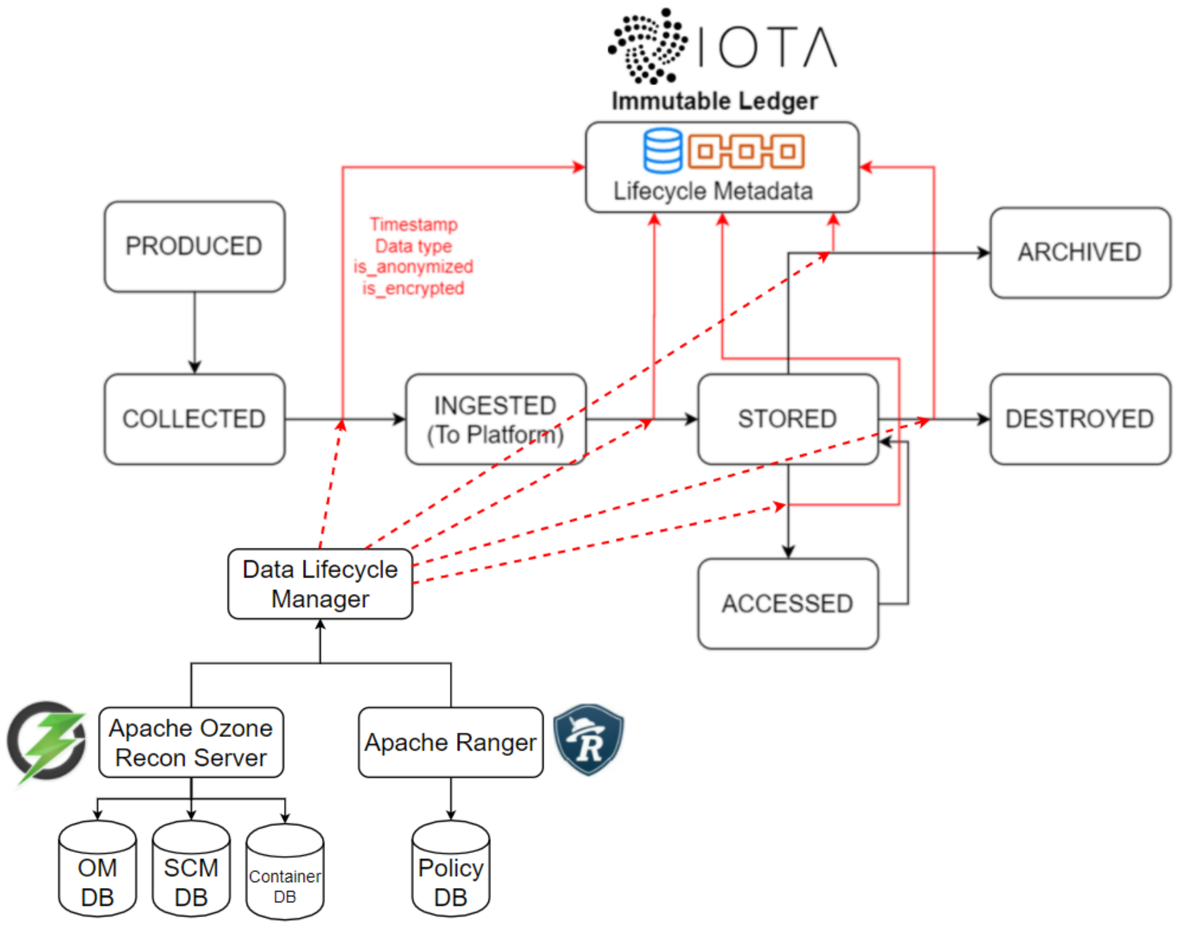


Figure 2.1 Data Lifecycle Manager Architecture showing integration with global filesystem and policy manager

2.2.2. Active Data Product (ADP)

An active data product is indeed a dataset encapsulated in a secure container that allows access via a well-defined access point that conforms to the terms of an agreed-upon contract. A data product is said to be active as it can operate in an external environment. It is indeed a self-contained, secure executable package that must be run in order to allow for the utilization of the data it contains. When requested by external agents to access the data, it will ensure that the request and the response comply with contract terms, usage, sovereignty regulations, and boundaries defined.

A contract definition language using YAML is under development, by which the data owner can define the terms and conditions for accessing the data. The contract is then enforced by the contract controller. Any legitimate access to the data will be recorded in a blockchain for contract term enforcement as well as for auditing and accounting. The ADP component is a prototype and may not be secure enough for production environments with high-sensitive data.

Each UC that aims at sharing data with other UCs or external systems/parties may benefit from this component.

The general user interaction with the ADP is depicted in Figure 2.2 below.

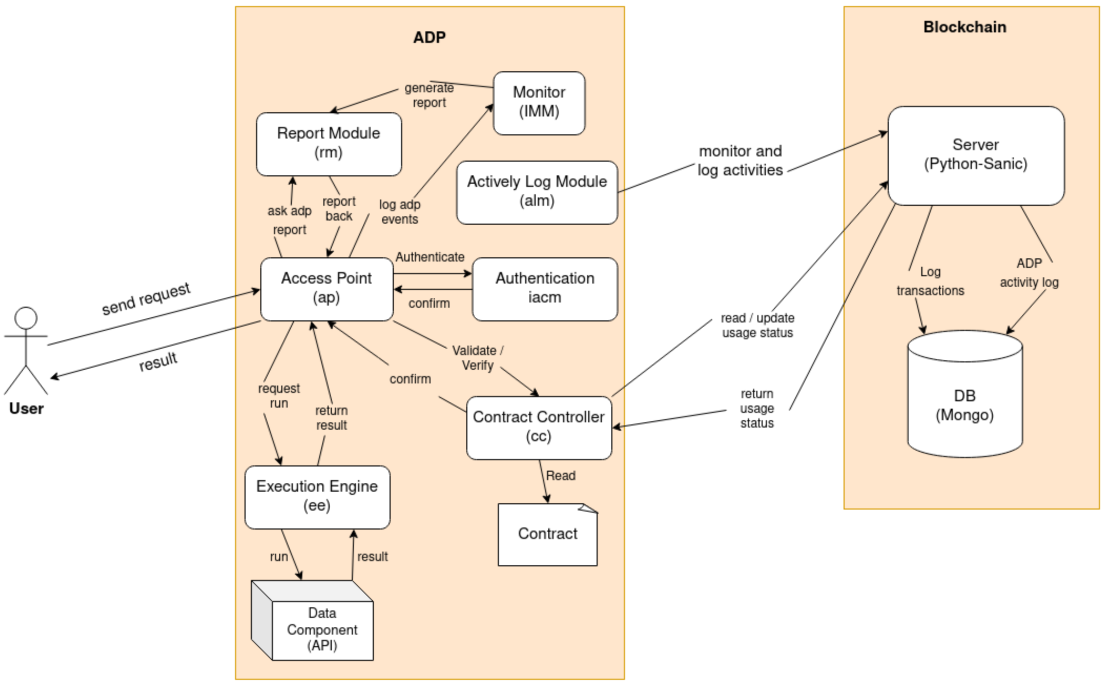


Figure 2.2 Interaction model of the ADP and a user (agent)

The ADP encapsulates a dataset (or an AI/ML model) and a contract that governs the access and usage terms of the dataset. When an agent requests to access data, it needs to provide the querying parameters. The access point first validates the request by checking authentication and authorization policies as well as usage policies and terms as in the contract. Therefore, if the request violates any usage of the terms, the request will be decarded and a cost associated with it will be submitted to the blockchain. This is to mitigate denial of service and data exploitation attacks. Any successful request will also be written to the blockchain for usage tracking. The access point validates the parameters against the contract terms. For example, an agent may be prevented from filtering data based on the gender of people in the dataset, or it can be prevented from requesting data of people under 18 years old. After parameter validation, the request will be passed to the execution engine to construct the result set from the dataset/model. The result set will also be validated against contract terms. For example, the input parameters may not explicitly ask for data about females, but the result may contain such data. If there is a restricting term in the contract, those records will be dropped from the final response. As both served and discarded resources are logged in a blockchain via the activity Loggin Module, usage can be tracked not only by the ADP instance but also by the parties of the contract as well as third parties. At the moment a prototype of the ADP has been implemented and is under test. The blockchain is simulated by MongoDB, which will be replaced after the contract controller reaches stable status. Research and development on securing the whole ADP against malicious agents and contract parties is ongoing.

2.2.3. Data Storage

The data storage solution in the BRAINE platform has certain key requirements. A BRAINE EMDC consists of multiple worker nodes each with a certain storage space available for applications and services. The EMDC may also consist of additional nodes specifically purposed to provide additional storage space. The nodes may comprise of different CPU architectures (ARM or AMD).

The goal of the distributed data storage solution is to configure all the available storage space as a single filesystem, creating a distributed pool of storage resources, such that they can be accessed by different workloads dynamically. The solution should seamlessly integrate heterogenous architectures. The solution should also allow configuration of smart, flexible and efficient data placement strategies, with the capability of dynamically moving the stored data across worker nodes on-demand, regardless of the type of underlying CPU architecture.

As the BRAINE platform is based on Kubernetes (K8s), the storage solution must be able to provide stateful K8s applications with a persistent volume (PV) to store their stateful data. Finally, the storage solution should: allow for security and encryption protocols to be incorporated; provide multi-protocol support for enabling access through multiple APIs and mechanisms and; allow integration of plugins for management of data including auditing, tracking and policy enforcement of events.

With the above requirements in consideration, various filesystems and object stores were reviewed and Apache Ozone was selected as the base to build the storage solution on. Ozone is a scalable, redundant, and distributed object store that aims to provide a resilient cost-efficient solution while providing features necessary to function effectively in containerized environments.

The distributed data storage in an EMDC is designed to run a single filesystem based on Apache Ozone. The Ozone management services including Ozone Manager (OM), Storage Container Manager (SCM) and Container Storage Interface (CSI) are containerized and created as Kubernetes (K8s) applications. Features that are required for BRAINE platform but were lacking in Ozone, were developed and the Ozone source code as well as deployment was upgraded accordingly. Two key such features are:

Interface for an external placement framework: One of the key components of the BRAINE platform is C4.9 Data placement framework. This component applies constraints and smart placement strategies to find the best nodes for placing a certain user data. To allow interaction with this placement framework, Apache Ozone's source code was modified and the relevant classes were updated to include API calls to the placement framework.

Support for heterogenous CPU architectures: Apache Ozone binaries available, as of the version 1.2.0, were built for AMD nodes. As the BRAINE platform also consists of ARM nodes, Ozone binaries and all the dependencies and supporting functions were built for ARM nodes as well. The updated solution can seamlessly run on a multi-architecture Edge device such as the BRAINE EMDC.

2.3. Implementation

2.3.1. Apache Ozone

The distributed data storage in an EMDC is designed to run a single filesystem based on Apache Ozone and the user or system applications that run inside the EMDC use Ozone storage class if they require persistent volumes. The Ozone components themselves are also created and managed by Kubernetes orchestrator of the EMDC. "D4.1 First report on the status of WP4" explains the mechanism of implementing and deploying Ozone in the BRAINE platform.

Furthermore, as explained above in Section 3.2.2, Apache Ozone was upgraded to include certain features essential for the BRAINE platform and use-cases.

For interfacing with C4.9 Data placement framework, the Ozone source code was modified. The data placement framework is built as an external API This API uses the available information for data placement. It keeps the record of the present state of data allocation on various data nodes. The API also has the details of the constraints on the data defined by the application. These constraints may be on node sharing, hardware

(Intel, AMD, ARM), selected data nodes, or some specific location, etc. All these details are stored in config (XML and properties) files.

The modified Ozone requests this API to get the data node list to store the input data. Based on the available information about this new data request (application, user, persistent volume, etc.) and the stored information about the current allocation the data placement API returns the list of data nodes to Ozone. Finally, Ozone uses these nodes to store the data.

For supporting heterogenous CPU architectures, Ozone docker containers were built on an ARM node and a multi-tagged docker manifest was prepared. This manifest consists of both ARM and AMD images within it and when instantiated on a particular worker node, the container runtime picks the correct image for that CPU architecture. Additional images related to CSI i.e., CSI node driver registrar and provisioner were also prepared in the same way with one image tag wrapped around ARM and AMD images. Finally, Kubernetes manifest files were developed that create K8s objects based on these multi-tagged images. In addition to the Ozone management components, one data-node component is instantiated on each worker or storage node that has to become part of the storage system.

2.3.2. Interfaces for accessing data

Various interfaces available to users or application of the BRAINE platform for accessing data are detailed in "D4.1 First report on the status of WP4". These interfaces include, Container Storage Interface (CSI), S3 Interface, Ofs and O3fs Hadoop Compatible Interfaces, Command Line Interface, Java API and a Recon Server. All of these interfaces are still supported for accessing data. Additionally, the CSI interfaces and components are upgraded to support heterogenous CPU architectures, as explained in Section 3.3.1 above.

2.3.3. Security features

Kerberos authentication

Apache Ozone uses Kerberos authentication to secure the cluster. Today to enable security in ozone cluster, we need to set the configuration *ozone.security.enabled* to true and *hadoop.security.authentication* to Kerberos in the core-site.xml file. Ozone uses the concept of tokens along with the Kerberos server to reduce the number pf requests to the Kerberos server. Once a client is authenticated, Ozone issues delegation tokens and block tokens to the clients. These tokens allow applications to do specified operations against the cluster, without having to request the Kerberos ticket for each operation. Its implementation and detailed analyses are still under progress

KMS

Transparent Data Encryption (TDE) allows data on the disks to be encrypted-at-rest and automatically decrypted during access. For Ozone, we can enable TDE at the key-level or the bucket-level. TDE is enabled at the bucket-level when a bucket is created. To use TDE, admin must setup a Key Management Server (KMS) and provide that resource identifier to Apache Ozone by configuring hdfs-site.xml. The property *hadoop.security.key.provider.path* in the hdfs-site.xml file is set to the KMS path. Once this file is configured for our cluster, then we can create the encryption key and enable encrypted buckets.

To create an encrypted bucket, the client needs to:

Create a bucket encryption key with hadoop key command line interface,

```
hadoop key create encKey
```

The command creates an encryption key for the bucket we want to protect. Once the key is created, it can be used for reading and writing data into a bucket.

To assign the encryption key to a bucket in Ozone,

```
ozone sh bucket create -k encKey /vol/encryptedBucket
```

After this command, all data written to the encryptedBucket will be encrypted via the encKey and while reading the clients will talk to Key Management Server and read the key and decrypt it.

2.3.4. Preventing cache attacks and micro-architecture induced vulnerabilities

Cryptographic mathematical functions in use are generally secure. It means that a simple set of encrypted data cannot be used to discover the applied cryptographic key in a reasonable amount of time, even when using the most powerful computer of the world. Cryptanalysis cannot succeed.

However, in some cases, side-information to the set of encrypted data, generally acquired during cryptographic computation, can turn the cryptanalysis mathematical problem into something that can be computed in a reasonable amount of time. This is called a side-channel attack.

There are many origins for side-channel leakage. A recently discovered one is induced by the hardware architecture of modern microprocessors itself. More precisely, it is due to branch prediction and other forms of execution speculation used to significantly speed-up computations.

This is very annoying because this kind of hardware architecture and techniques are present in all modern microprocessors. And permanently disabling the excursion speculation has a huge impact on performance.

The more efficient path is to modify or fix part of program where it is analyzed that information leakage, being a danger to security, occurs.

The Catalyzr tool, developed by SIC in Braine, is here to assess the security of software with regards to side channel attacks. The goal of the tool is to analyze programs in order to detect, locate and characterize different types of vulnerabilities associated with those attacks.

The tool operates in a semi-automated way, allowing the user to generate a vulnerability report given an input software target. Depending on the type of vulnerability, the Catalyzr performs the detection either by static analysis, by identifying leakages at the source code level, or by dynamic analysis, by executing the program binary and discovering vulnerabilities from its observable behavior.

In the context of Braine, the targeted vulnerabilities are microarchitectural vulnerabilities, and are detected by static analysis. The Catalyzr prototype for Braine usage is working, has been tested, and has now started to be used in “real case” application on some Braine programs.

Semantic Web

The idea is to apply AI to the semiconductor supply chain and supply chains that contain semiconductors. So far, the development of this idea is too complex, because it requires individual exposure and pattern recognition. AI is already being used in other areas such as customer order behaviour prediction, over- and under planning, and wafer maps. It will be better to apply the idea first to an easier understandable domain.

3. Efficiently handling data at the edge at scale

3.1. Secure and Optimized Data Placement

Data stored in the storage system of a BRAINE EMDC may belong to different data owners, users, and applications. Categorizing data and storing it separately will assist in maintaining data privacy and security. For instance, data belonging to a different application or data owner can be stored on different data nodes guaranteeing isolation among users and their access requests.

A data placement framework is supposed to control the data/ file placement in the storage system based on resource requirements, privacy and security constraints, and performance requirement (to optimize). This data placement API is implemented as external API out of Ozone file system. The modified Ozone will call the placement API to get the suitable nodes for the placement of particular request. The design of such a placement framework will be based on:

- User's/ Application's data placement requirement. These requirements mainly provide the details about storage constraints to the data placement API. This information may be defined for each application as input from application or the information about the application in the system (like persistent volume created inside the Ozone). These requirements can be stored as metadata of the input file and used for data placement. If there are no constraints defined on some application's data, then its allocation will be done with the Ozone's default placement policy.
- The data placement API also keeps the present state of data allocation on various data nodes (allocation of application's data on different nodes). The placement API uses this record and the input data requirements to identify the data nodes suitable for secure data placement.
- A modified Apache Ozone that calls the external placement API to get the selected data nodes to store the input data.

3.2. Efficient analysis of the time series and detection of motifs

Motif Discovery tool (MOD) is an advanced analytic tool specialized for time-series data streams. Such a stream cumulatively represents a considerable load on network bandwidth, which is very demanding in edge-to-cloud communication. When most data processing is offloaded to the edge, MOD significantly reduces the communication load.

Support modules are required to achieve such a communication relief: The Discovery module and the Detection module.

The Discovery module analyses the batched offline time-series data to find repeating patterns (motifs) and builds a dictionary of distinctive motifs shipped to the cloud. The Detection module then consumes an online stream of sensory data. It converts segments of detected motifs into discrete references in the dictionary, and only the references are transmitted to the cloud. In the cloud, Digital Twin (DTwin) consumes the references and updates the production model accordingly. The overview diagram is depicted in Figure 3.1.

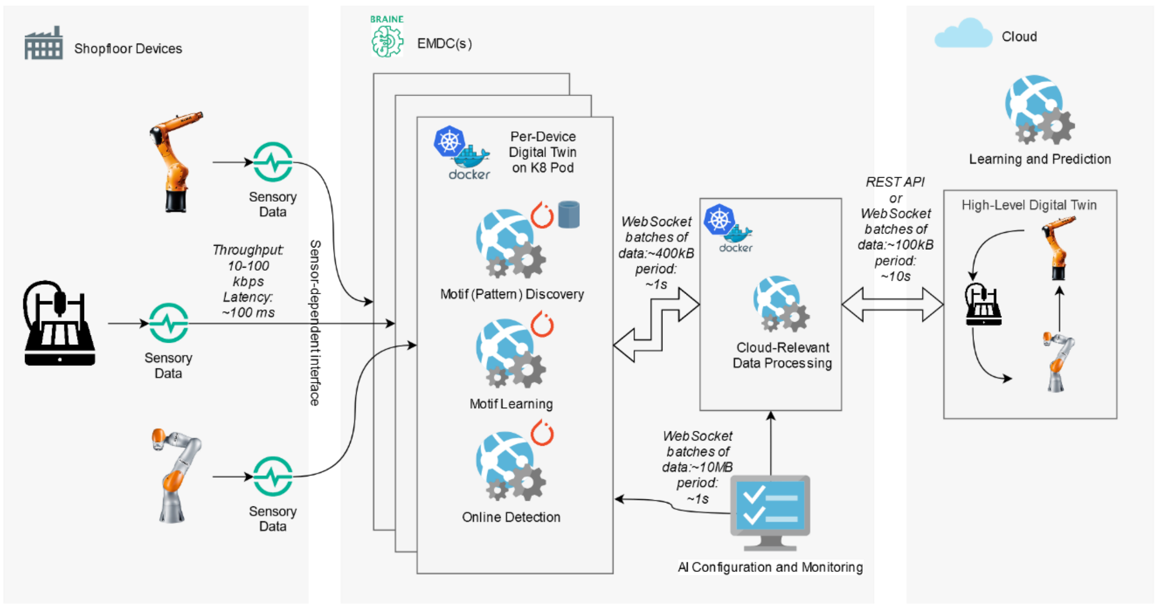

Shopfloor Devices
Sensory Data
Throughput: 10-100 kbps
Latency: ~100 ms
Sensory Data
Sensory Data
Sensor-dependent interface
BRAINE
EMDC(s)
Per-Device Digital Twin on K8 Pod
docker
Motif (Pattern) Discovery
Motif Learning
Online Detection
WebSocket batches of data:~400kB period: ~1s
docker
Cloud-Relevant Data Processing
WebSocket batches of data:~10MB period: ~1s
AI Configuration and Monitoring
REST API or WebSocket batches of data:~100kB period: ~10s
Cloud
Learning and Prediction
High-Level Digital Twin

Figure 3.1 The overview of the whole MOD tool design

The development of the Discovery, Learning and Detection module reached the release point. Those modules are now at the alpha versions and are tested in a CNIT Braine Testbed and on real HW at CTU Testbed for Industry 4.0. Currently, these components are being integrated within the UC3

3.2.1. Motif Discovery Module

The Motif Discovery module enables the discovery of all repetitive patterns of any size in any time-series data. The time-series data from the perspective of BRAINE are sensory data from machines and devices on the shop floor. The discovered patterns represent the unique operations of the machine.

The Discovery module requires access to the Influx database that collects and stores time-series data from sensors. Within the Influx database, there has to be a separate bucket dedicated for online storing the sensory data stream and providing those data in batches to the Discovery module on demand. The design of the data processing pipeline is depicted in Figure 3.2.

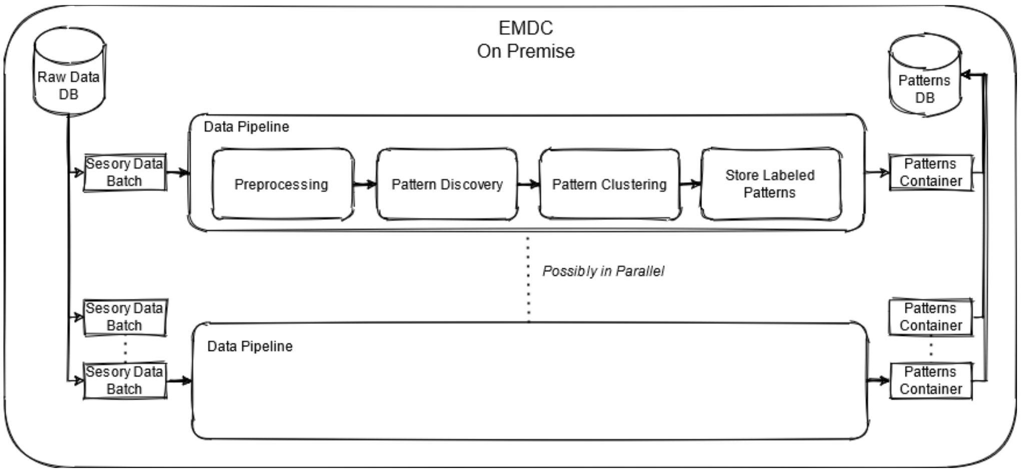


Figure **3.2** Discovery module data processing pipeline

The discovered patterns are required by the Learning module (WP3-T3.1) to train the Detection module (WP4-T4.2) properly

3.2.2. Detection Module

The Detection module of the MOD application uses state-of-the-art machine learning models to detect the current operational state of the machine. This module uses online time-series data as an input and returns the current operation of a device. Additionally, the Detection module checks the incoming time-series data for deviations from its typical behaviour. When a deviation occurs, an alarm is set on, and subscribers are notified. The design of the data processing pipeline is depicted in Figure 3.3.

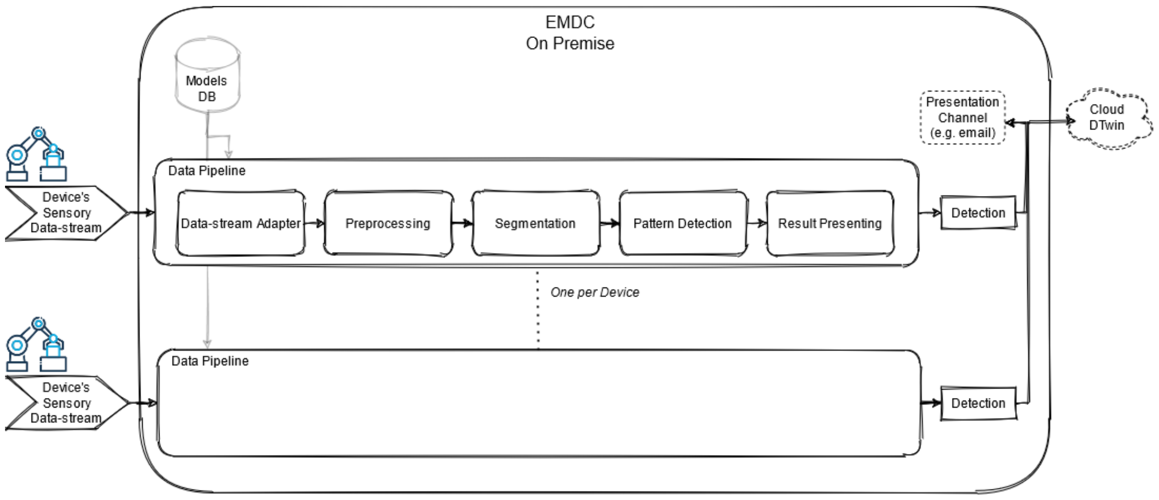


Figure 3.3 Detection module for online pattern/motif detection in data processing pipeline

The detection module requires access to a notification channel and data sources delivered by the Service mesh component of the BRAINE platform. The service of the Detection module is utilized in UC3. The KPI KP1 in chapter 3.3.5 in MS5 indicates the Detection modules' achievements.

3.2.3. Digital Twin Module

Digital Twin (DTwin) module is designed to run in the cloud. It provides a high-level overview of the entire factory manufacturing process in the form of a discrete-event system. It requires a dictionary of learned models supplied by the MOD Learning Module (WP3-T3.1) to reconstruct the continuous data streams from the sensors when needed. Additionally, it requires online detection of current machine states provided by the MOD Detection Module (WP4-T4.2).

3.3. Virtualized RAN and use-cases cooperation

To address the requirements of data handling at scale in BRAINE we should look at vRAN and use-cases from the perspective of further alignment of the two. The alignment planes are two: (a) resource optimization at vRAN (radio, computing) and (b) semantic alignment of the vRAN domain and a use-case domain, that targets contextual understanding of use-case's data characteristics, topological as well as HW information about data sources/targets, existing options and variants to achieve optimization etc.

The vRAN requirements for the WP4 revolve around two aspects, which are essential for pursuing the *tighter integration between* vRAN and the BRAINE architecture. These are especially (i) the potential that vRAN can introduce to the relevant optimizations of data handling by means of appropriate resource management and configuration of the slice and (ii) the contextual knowledge about use-cases and its requirements. The key perspective considered here for such integration (or coordination) between 5G workload and underlying EMDC architecture, is effectively handling data at scale. Handling data effectively at scale maps to addressing performance requirements of any BRAINE workload, and especially the secure 5G wireless communication infrastructure.

At the current stage of the BRAINE release 1.0 the use-case characteristics and requirements have been identified and provided in Deliverable 4.1, "First report on the status of WP4". Also, the technical requirements of vRAN to operate on top of existing EMDC HW, have been described in Deliverable 5.1, "Functional requirements for BRAINE infrastructure and platform service". But so far there was no direct fusion and alignment of the two at this stage considering either semantic, or technical perspectives. So far, the relation between vRAN and the BRAINE use-cases (especially UC2) has been performed at the level of capabilities identification, that is indicating the characteristics of a use-case and the requirements the need to be fulfilled at 5G connectivity level in order to deliver it successfully.

In this milestone report we provide a targeted approach to address high alignment between use-cases and optimizations at vRAN level – the target is to seek redundancy removal in the area of data volume, as well as number of requests for resources.

Below, the two essential approaches to prepare vRAN for data handling at scale are identified, namely (a) baseline approach and (b) extended approach. The former relies on appropriate configuration of vRAN instance(s) while the latter builds on top of semantic integration between vRAN resource management (radio, computation) and use-cases.

3.3.1. Baseline approach for data handling at scale with vRAN

There are multiple means that can support handling data at scale in the edge EMDC when it regards vRAN as a special purpose workload. The vRAN besides the EMDC scheduler and orchestrator, influences data handling capabilities from radio/computing resource perspective. There are at least a few options for special purpose configuration of 5G vRAN, which depends on the services characteristics:

1. **Option1**: Service provisioning for a mMTC service (slice) - in the baseline approach vRAN requirements can be limited to the appropriate identification of service type (or types) that a particular use-case represents. Basically, in 5G there are three types of services supported: eMBB, URLLC and mMTC. Nominally only the latter one is considering "handling data at scale" by design (special mechanisms to decrease signaling overload), which considers the IoT/IIoT based use-cases (e.g., eHealth, smart city). The typical mMTC use-case maps to a set of requirements that manifest mainly in a large number of terminals, with low but bursty traffic demand. Usually, the data is sent in uplink as the UE devices represent some sort of sensors. There are some special features foreseen by 3GPP on the side of vRAN that are provided to assure high robustness like e.g.: *grant-free signaling*, *NOMA* multiple-access and so on. This way the network which has been configured for mMTC is already provisioned for the requirements of multiple connections.
    a. Baseline methods to support data handling at scale by vRAN configuration consider:
        i. Slice provisioning and configuration to address QoS/QCI profile of a use-case application traffic
        ii. Admission/congestion control, scheduling solutions tailored to a use-case requirements
        iii. orchestrator based tuning of workload placement, scaling that supports computing resource optimization (e.g., OPEX driven)
    b. Comments:
        i. The essential aspect that determines the optimization capabilities is the vRAN disaggregation granularity. Namely the higher the disaggregation level the more capabilities for vRAN optimization and tuning (regarding the computing resources usage) due to more flexible adaptation
        ii. such optimizations are subject of task T3.3 in BRAINE and they address aspects of vRAN workload modeling, prediction and resulting placement (scaling) decisions.
        iii. this option has the capability to fine tune both radio and computing resources to adjust it to the needs of a use-case.
2. **Option2**: Establishing local breakout by means of vRAN - MEC cooperation (for URLLC, eMBB) - alternatively in case when data demand has characteristics that demands e.g., very low delays (real-time behavior) then we speak about URLLC

type of applications, that demand ultra-low latencies and very high reliability (i.e., very low tolerance for packet loss). In the latter case we can be speaking about grant-based communication (so legacy signaling) but there can be mechanisms like ETSI *MEC servers* configured in the network in order to offload data-handling from cloud to the edge. This would result in lower delays, higher offloading capability and thus also could result in less load towards the network core.

3. **Option3**: Introduce vRAN resource pooling - however with virtualization of RAN (vRAN) and opening of RAN interfaces (open-RAN), one faces the new capabilities for pooling “core/CU/DU” resources in the edge or edge cluster servers. This way the scaling of vRAN becomes feasible both: (I) horizontally i.e., at the edge, or also (ii) vertically moving some elements of 5G processing infrastructure workloads to cloud (or multi-cloud). Such scaling flexibility is being addressed in task T3.3 in combination with workload prediction and placement algorithms for vRAN.
4. **Option4**: Acceleration of vRAN radio stack components - data handling at scale is tightly related to the capability of “acceleration on demand” that is controlled at edge orchestration in the process of slice (or network) design and provisioning. Acceleration can utilize special purpose HW resources, which can be made available to a radio stack, and thus offload the CPU processing of the entire EMDC. It is essential to notice that efficiency of such performance boosting will heavily correlate with the level of vRAN disaggregation as well as particular layer considered, a specific mechanism to be accelerated, and so on.

To summarize, this baseline approach calls for targeted deployment of vRAN that is service aware and tuned with special use-case driven settings (e.g., grant-free access) or additional mechanisms, to serve data handling at scale (Option1-4). The vRAN itself brings the possibility of scaling own resources in case user density, traffic volume to be processed is higher. This approach can further be combined with special purpose resource management for vRAN – described below as “extended approach” to data handling at scale. In BRAINE release 1.0, Option1 will mainly be key focus point for configuring the 5G infrastructure during integration phase. Later, towards Release 2.0, the options Option3 (related to T3.3) and Option4 (related to T2.2) are considered crucial. In turn Option1 can be combined with optimization objectives identified in T3.3 (Option3), to support selected resource management mechanisms.

3.3.2. Extended approach for data handling at scale with vRAN

For the so-called extended approach to 5G secure vRAN provisioning, that targets handling data at scale, it is assumed that by incorporating contextual information about the use-case into the vRAN resource management processes is essential. The data handling at scale can be optimized by means of wise decisions regarding resource allocation – such that allows for smart adaptation of infrastructure and the relevant signaling.

In order to identify the synergies and cross-relations between the use-cases on top of the BRAINE platform and the 5G network component and the complementary requirements i.e., of hosting the 5G workload on an EMDC, it is essential to provide means that would allow capturing and better defining existing dependencies between the two. Such means in BRAINE can naturally be: (i) BRAINE ontology (ii) common telemetry framework where all the metrics can be fused/filtered by any consumer around Kafka bus (iii) services and use-cases ontology or (iv) use of corporate memory tool (cmem) that provides data exchange options at scale. The architectural building blocks of BRAINE, like: telemetry collection, metrics semantics unification, data sharing by message bus, are causing a situation where BRAINE becomes an alternative to using MEC. However, MEC as a set of specifications seems less flexible and demands purpose-build adaptations and extensions. As such BRAINE brings homogenized system that allows cross-layer optimizations.

Based on the goals of "handing data at scale" the BRAINE EMDC could benefit from the tailored vRAN design in the following ways:

- **Option1**: vRAN own optimization combined with the complementary adaptations on the use-case side (including SW or HW)
    - Methodology
        - vRAN is capable of accessing the use-case's contextual KPI/metrics, which are exposed on the monitoring platform of BRAINE (e.g., Kafka bus).
        - It becomes possible to define a solution on the side of vRAN (algorithm, xApp) which will be able to support the data-at-scale handling by e.g.: (a) preventing selected data flows from accessing the network in UL or DL direction (b) limiting selected data flows by

        considering various mitigation strategies as well as semantic inference about “multi-modalities” of sensors used in a UC (i.e., use acoustic sensor when a location is empty and some cameras can be turned off)

        - special purpose BRAINE/vRAN controls need to be defined to interact with a use-case actuators/sensors at the communication network level
        - special purpose BRAINE/vRAN controls can consider either the (a) model-based control system available to vRAN (e.g., SDN based RAN controller) that actively manages the use-case elements (e.g., cameras) and its connectivity layer or (b) semantic driven and policy-based mechanism by which it is possible to cross-relate and orchestrate vRAN together with a use-case (here the role of SDN RAN controller is also possible) or (c) dedicated SDN RAN controller xApps.
    - Insights
        - The benefits here are related to the more proactive, but also application (user) agnostic, bottom-up adaptations of a use-case elements (e.g., camera, sensors, robots, etc.) by means of a provided knowledge base (e.g., SUMO ontology) encompassed with relevant rules and query capabilities.
        - <u>Semantic model of a use-case can also improve the accuracy of prediction at the vRAN resource management level</u>. Semantics allows more insights into the characteristics of use-case traffic flows, its evolution in time to reflect changes in use-case operation and so on. Moreover, it is relatively easy to combine semantic knowledge with policy-based rules and system state based on up-to-date monitoring system telemetry.
- **<u>Oprion2</u>**: Holistic workflow based, ML/AI driven, cross-optimizations of vRAN and use-case elements (SW and HW components) that are rooted in the availability of high-level workflow definitions that address both vRAN and the particular use-case operations.
    - Methodology
        - This option brings the most of capabilities of cross-relating vRAN (infrastructure) with the use-case (applications) but it can be

        already done by mechanisms present at the workflow lifecycle of both vRAN and use-cases.

        ▪ Here the relations between both workload types, as well as related HW elements, are happening at the level of processes definition, especially considering the relations between ML/AI models, their similarities, goals (utility functions) and so on.

    o Insights

        ▪ This approach should be combined with the underlying policy-based subsystems at BRAINE EMDC for maximized performance. Moreover, the careful design of ML/AI workflows will be crucial.

It needs to be noted that different options presented above for the so called "extended approach to data handling", naturally map to the differences in the integration (coupling) level between vRAN (5G) and the surrounding use-cases and its components. The higher the option number, the higher the capabilities for cross-workload interactions and especially the influence on handling (predicting, mitigating) data at scale can be foreseen. It is assumed that by introducing use-case characteristics to the modeling of optimizations at the vRAN side - the benefits for data-handling can reach the highest levels. Here the strong requirement to proceed is the design and filling of the use-case specific ontologies with related instances and SPARQL rules to infer from ontology contents. In BRAINE release 1.0, the use-case ontologies do not yet exist, this is why the topic needs to be further elaborated towards release 2.0.

3.3.3. Impact on design and implementation

In order to allow addressing the challenges resulting from abrupt (or foreseen) data volume increase at an EMDC, there should be provided higher level of semantic cross-relation between the use-case characteristics and the radio/computing resources behind vRAN within the EMDC. The summary of the impacts influencing both sides (vRAN, use-cases) has been presented below. It needs to be highlighted that the aspects considered in this D4.2 require that D3.4 (monitoring of metrics) is first accomplished to then build on this and evolve towards D4.2 concepts.

By allowing the vRAN to better "understand" the role and criticality of the data foreseen to be delivered by use-cases of BRAINE EMDC (in both UL and DL directions), the semantic contexts of the two aspects, namely: use-cases and vRAN, needs to be considered. In this way a weak coupling can be provided so that vRAN decision making (resource allocation, congestion control, etc.) can be linked to the actual goals and current state of the use-case operation (including its underlying sensors/actuators). Here the common denominator would be at the level of semantic bridging (e.g., by means of shared vocabulary and rules).

The baseline approach is that a use-case is represented by the type of traffic profile in order to determine appropriate slice requirements. But such approach is not enough to let the 5G infrastructure engage in adjusting (by e.g., RAN intelligent controller) to the changing environment conditions and especially to the changing execution context of use-case operation (considered here broadly as a complete OTT system including SW and HW components). The approach considered here is to an extent motivated by the physical layer security research (i.e., PLS), where the influence on the resource allocation regarding an eavesdropping user terminal, has the capability to radically reduce their vector of attacks.

3.3.3.1. Impact on design of BRAINE architecture

The BRAINE architecture allows inclusion of semantic description about use-cases, moreover various policies can be defined for and EMDC node operation under changing context of execution. This way certain steps will be done for the integration stage planning, in order to assure the following targets:

- Selected use-case providers together with vRAN developers will co-design purpose-build **higher level KPIs** aggregating the details characterizing the use-case and enabling the understanding of use-case traffic flows - allowing for capturing cross-dependencies between space/time dimensions of a use-case at the level of networking infrastructure. This way technical means for aligning the two domains would be assured (i.e., applications and networked EMDC 5G infrastructure), by providing semantic indications of e.g.:
    - use-case traffic characteristics (QoS profile, requirements, …) - to understand what the traffic flow parameters and volumes are
    - number of HW elements used by a use-case as sensing devices (or actuators) together with their capabilities (e.g., HW, SW, I/O) and cross-relations to be able to capture that e.g., noise sensor can be orchestrated instead of a camera in place where there is low human traffic in order to identify the crowding situations

    - high level targets for data collection/generation (i.e., how devices acquire data, what data is actionable, when the data can be treated as “successfully delivered”, what data can be omitted in case of resources shortage)
    - geographical context of the services and HW elements - i.e., where the sensing/actuating devices are deployed, how much they are correlated in space and time between themselves and other elements of a use-case (or use-cases)
    - meta-description of models backing up vRAN and a use-case, together with ML/AI modeling pipelines that are used to train, validate, and update the models (or lifecycle)
    - alternative data sources identification i.e., various alternatives for data-fusion that may exist but could provide alternative sources of data to make use-case valid even in case of some performance degradations. Example would be to utilize “noise sensors” instead of cameras where “suitable”. Suitability should be provided in form of a rule or other kind of model.
    - The above parameters can also be considered in the process of learning mutual correspondence and cross-relations.
- Designing vRAN deployment will consider capabilities for intertwining the **slice design** (network and service descriptors) with the BRAINE semantic descriptors and especially service (use-case) ontology. Slice design represents the process of adjusting vRAN configuration to the needs of a use-case (traffic profile)
- The use-cases semantic description will be **capturing traffic flows/demands/statistics** with the indication of service requirements for successful SLAs and where/when data can be accommodated by the network (in which cases, for which HW, for which scenarios, etc.). Statistical models of the traffic (and variants thereof) should be able to be built and provided towards the orchestrator (workload predictor / resource manager).
- **Modeling cross relations** between workloads (e.g., a BRAINE use-case vs vRAN) will be performed in the integration stage, by considering optimal alignment of multiple layers:
    - Semantic models of use-cases as well as the network infrastructure
        - Application oriented
        - Security oriented
        - Resource oriented
    - Policy based network management mechanism and policy unification

    - Deconfliction of mutually excluding goals of various components (infrastructure, use-cases, management)
    - ML/AI models meta-descriptions with workflow descriptions

3.3.3.2. Impact on implementation of BRAINE releases

The BRAINE releases will depict various levels of maturity of vRAN, enhanced in order to handle data at scale. For release 1.0 the following aspects are considered.

The vRAN is provided as container-based Kubernetes cluster including: core network, centralized unit (CU), distributed unit (DU) as well as physical layer. At this stage the "baseline approach" for data handling will be prepared and the mechanisms for resource management will mainly be configured during the integration stage to suite the use-case characteristics. Whereas for the BRAINE release 2.0 another set of implementation steps is planned to match the "extended approach" for data handling. Namely the following features will be considered:

- vRAN orchestration and resource management demand to have <u>access to a semantic description of a use-case</u> (at the most relevant levels of generalization)
    - There should be KPIs/metrics together with rules that regulate their relation to underlying infrastructure of use-case infrastructure including sensors/actuators and network requirements
    - Specifications to be considered:
        - ETSI ISG CIM (context information management)
- Same way vRAN metrics/KPIs to be provided for the use-case elements (and decision-making process as such) directly or via BRAINE ML/AI workflow components (proxy mode)
    - Solutions considered
        - MEC APIs
        - RIC xApps
        - Prometheus exporter.

Optionally, in release 2.0 the following aspects will be considered, depending on the completeness and shape of the ontologies of use-cases:

- Assurance of security requirements for a use-case to highlight not only performance but the relation of performance and security goals (and trade-offs) of

  a use-case (this can utilize use-case workflow definition, or the use-case models workflow / pipeline)
    - Ciphering, encryption standards required by use-cases are potentially E2E so transparent to vRAN
- Availability of higher-level policies for BRAINE architecture with common semantics and syntax (i.e., based on the IETF specifications). Moreover, the policy selection, validation and enforcement will be carefully planned at overall level of BRAINE as well as aligned towards the “per component” policy subsystem (e.g., governed by the COPS or XACML models)
    - Implementation consideration
        - Identify suitability (and gaps) of BRAINE policies
        - Identify suitability of XACML/COPS
- The design of vRAN optimizations will consider the **use-case ML/AI models** (and meta-models) for inclusion in the process of e.g., prediction algorithms design (e.g., by knowing the use-case specific – measurement based – model of operation vRAN can be tuned to respond to the realistic traffic demands
    - Implementation consideration
        - Identify common ML/AI workflow tool (Argo, Kubeflow, etc.)
        - Identify metadata and other tools to allow cross-workflow alignment between vRAN and a use-case
- vRAN will be tailored to **access the use-case metrics and KPIs** at various levels of details by means of BRAINE common data bus (e.g., “number of cameras installed in location ABC”, “number of cameras that don’t recognize any person”, “number of cameras that can be turned off if noise sensor is in place”, “camera battery level”, type of surveillance scenario that is applied for the data collection and processing, etc.)
    - Implementation consideration
        - Identify proper interface to collect data from BRAINE (Telemetry platform) and make it accountable to vRAN
        - Identify relevant ontology information (classes of interest for KPIs, SPARQL queries, etc.) - or possibility of acquiring data with the help of the cmem[5] tool
- vRAN should be capable by release 2.0 to interact with elements of BRAINE architecture that can be used to search for use-case optimizations and traffic (data) compression or rationalization in certain situations.
    - Existing policy management system

- Existing resource management of the NFVI

The above considerations present set of concrete guidelines that should streamline the further work of WP4 partners and liaising with the WP5 use-case partners in order to provide semantic models that will be able to cross-reference with wireless-network ontology. The next update will be provided in the D4.3 document.

AI application profiling

The execution of complex Deep Learning (neural network) algorithms on heterogeneous hardware configurations opens the field for several optimization opportunities. However, the combination of large number of algorithms and hardware configurations makes the development of one-size-fits-all heuristics complex. In order to explore the optimization space and ensure optimal hardware utilization, thereby reducing energy requirements and improving runtime performance, we designed a profiling tool that can reconstruct the dynamic direct acyclic execution graph of deep learning algorithms.

This tool is integrated as part of the NEC's SOL compiler/runtime tool-stack, and it extracts computation graphs resulting from the tracing of a neural network execution, as shown in Figure 3.4.

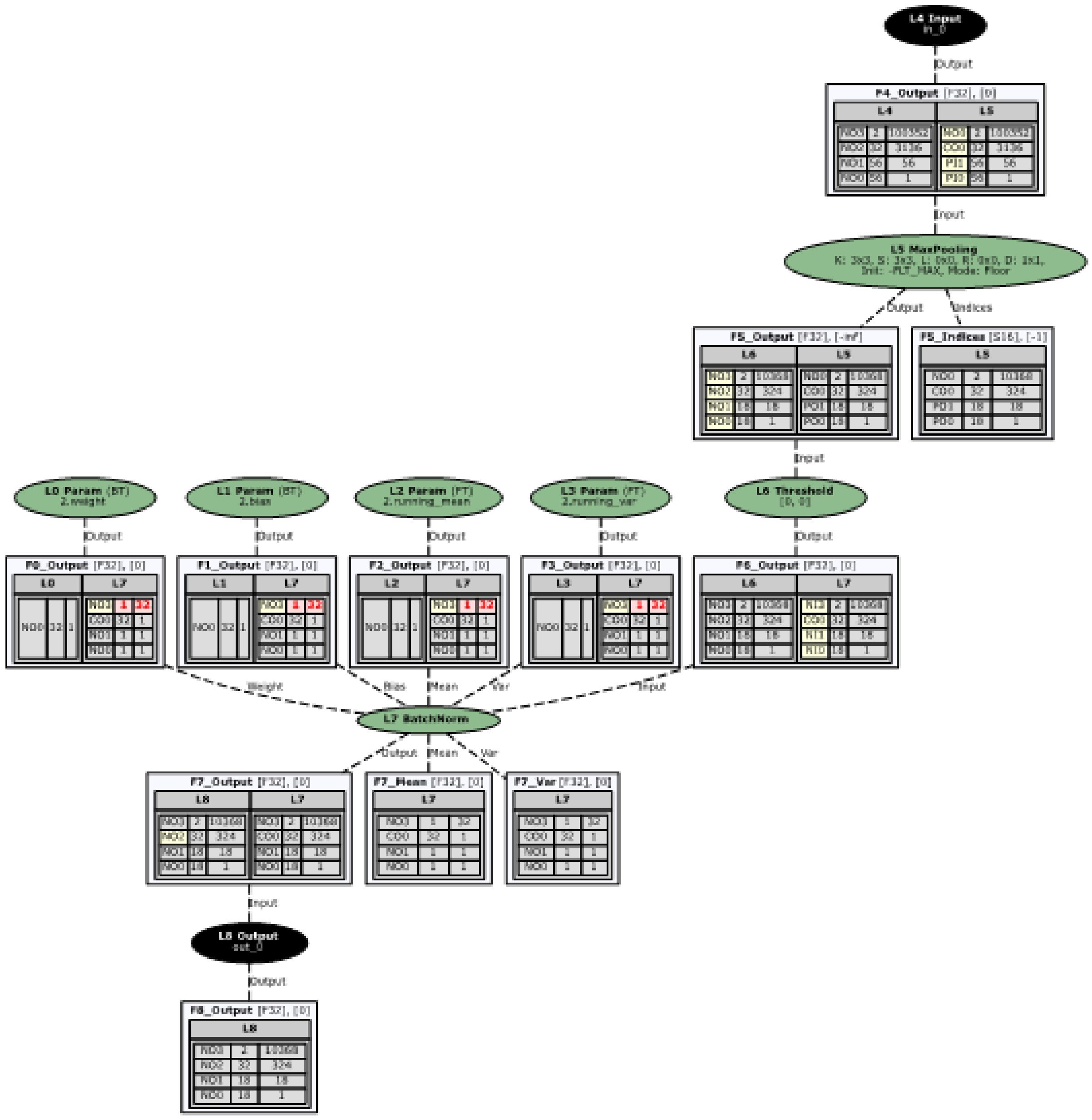


Figure 3.4 Neural Network execution profile extracted by the SOL runtime profiler

The extracted profiles are then ready to be delivered to the remaining part of the tool stack to apply optimization rules generated by potentially different optimization technologies, including heuristics and data-driven approaches.

The overall tool is integrated into applications by leveraging direct import in high-level domain-specific machine learning frameworks, such as PyTorch and TensorFlow, and therefore it can be shipped jointly with the application deployment units.

One of the additional features being currently introduced in the framework is the ability to perform short micro-benchmarks on target hardware executors (which might be made available both at development or operational time). These executions are used to measure the performance of certain parts of the input neural network, in order to better profile the execution taking into account the specific nuances of the target hardware.

Integration Plan

The AI profiler is developed as part of a compiler stack that can be used during the application development time, integrated with common AI frameworks such as TensorFlow. We have integrated the technology in a container template that can be used as basis for the development of data analysis applications.

Therefore, in terms of integration the AI profiler is considered as part of the AI analysis application’s software stack, and it operates transparently from any other EMDC's subsystem. However, its general life cycle is handled like any other container/service deployed on the platform.

4. Workflow definition language and authoring tool

Workflow definition language The Workflow definition is currently being handled by the OWL-S process class (**Figure 4.1**) which allows the specification of Atomic, Simple and Composite processes as well as their parameters and results. Simple Processes are single step processes and are realized by an Atomic Process. CompositeProcess are composed of two or more processes and have their execution specified into constructs such as Sequence, Split, Any-Order, etc.

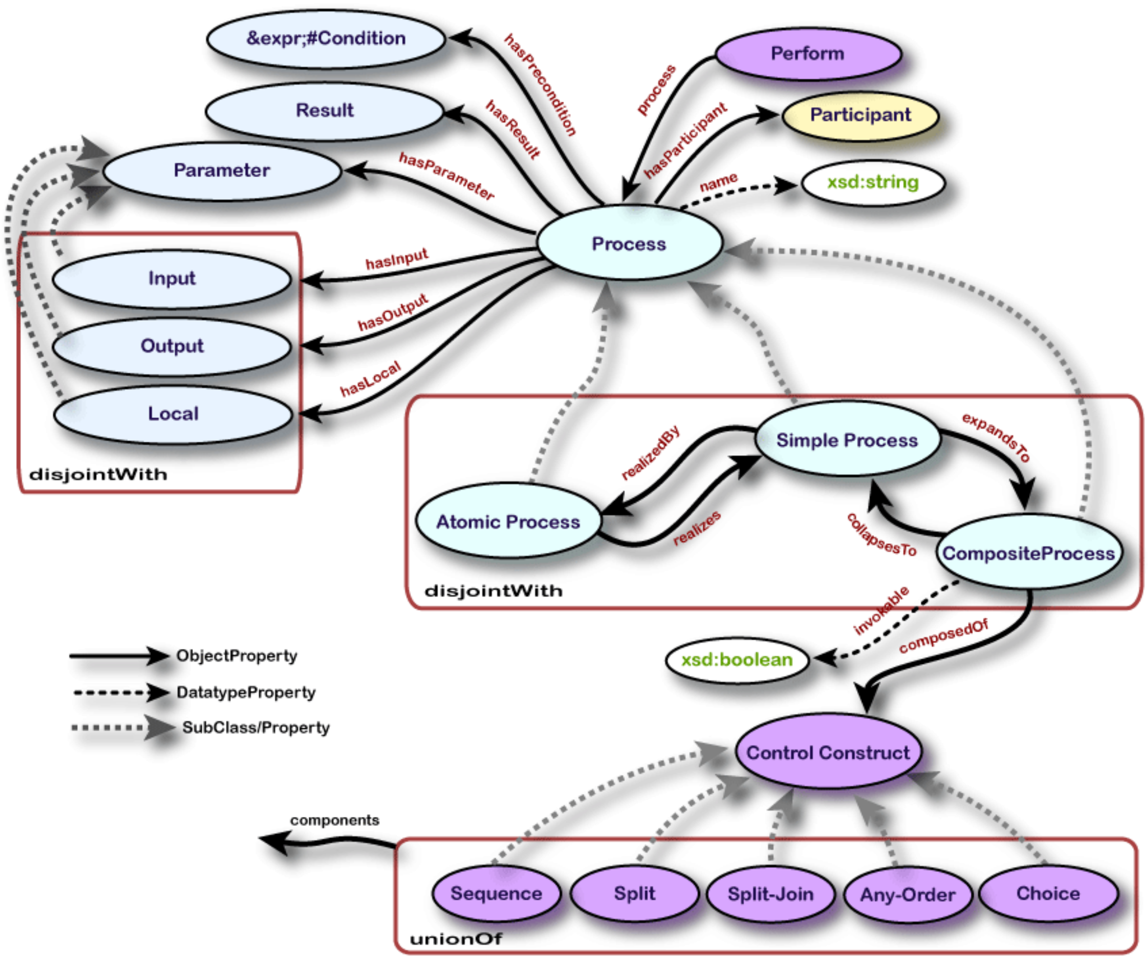


Figure 4.1 Service Deployment Specification Window

For being open source and tightly integrated with Kubernetes, Argo was chosen as the service workflow execution framework. In addition, it offers all required functionalities in the project scope and users can describe workflows in a declarative way using manifests in a similar fashion to those of Kubernetes and Docker.

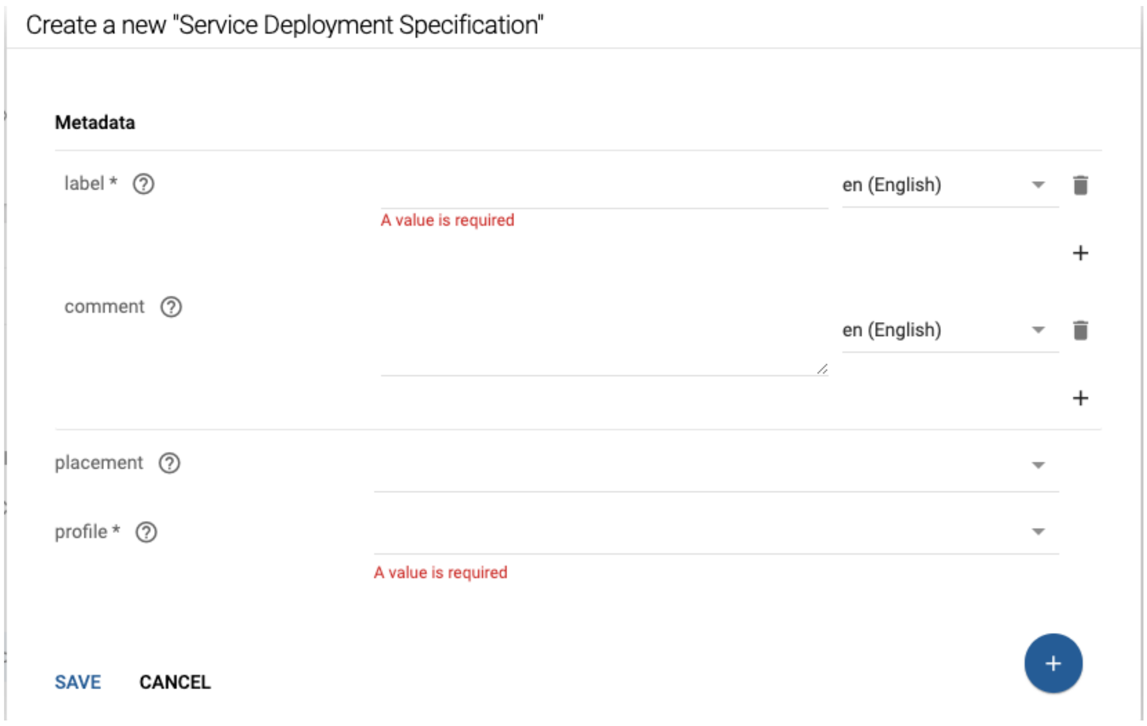
With the addition of the Argo framework to the BRAINE software stack, we add two formal ways for workflow definition (1) through OWL-S and (2) Argo Manifest file (Listing 4.2).

The latter, however, seems to be more aligned with the overall project architecture and can be easily managed by the user as well as by the system.

```
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
  generateName: hello-world
  labels:
    workflows.argoproj.io/archive-strategy: "false"
  annotations:
    workflows.argoproj.io/description: |
      This is a simple hello world example.
spec:
  entrypoint: hello-world
  templates:
  - name: hello-world
    container:
      image: hello-world
```

Listing 4.2 Argo hello-world workflow example.

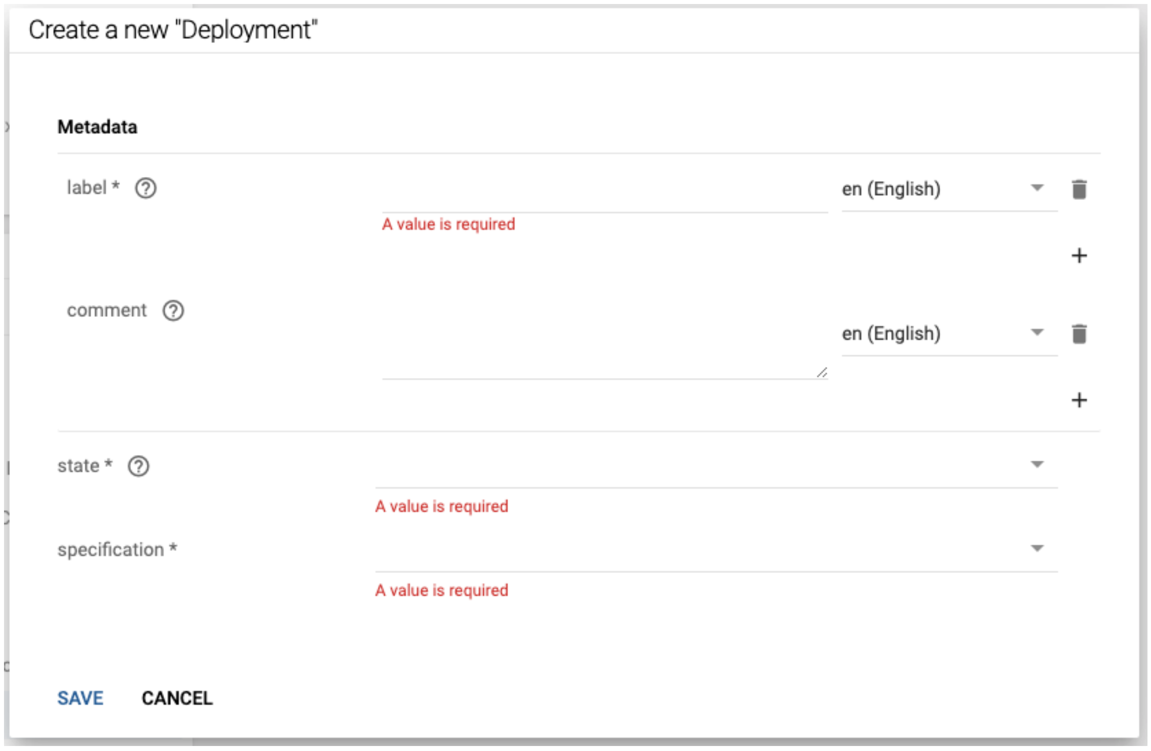
Authoring tool on its current implementation, the authoring tool allows the definition of the ServiceDeploymentSpecification by selecting the desirable placement (Node) and ServiceProfile. The ServiceDeploymentSpecification is a conjunction of the ServiceProfile containing metadata regarding the service, and the placement, containing information about the Node. This schema allows the reusability of different Specifications for (re)deployment. However, there is no possibility to define composed Services (Processes). This additional information is part of a discussion that is still going on regarding the definition of the input-data as well as the service itself. One option is to use the OpenAPI specification for defining parameters, or extend the existing OWL-S. Either way, it will be required to extend the metadata for defining the input metadata. **Figure 4.3** and **Figure 4.4** show respectively the Service Deployment Specification and Deployment User Interfaces. We intend to explore the creation of interfaces in other front end development frameworks in the next period.


Create a new "Service Deployment Specification"

Metadata

label *	A value is required	en (English)	
comment		en (English)	
placement			
profile *	A value is required		

Figure 4.3 Service Deployment Specification Window

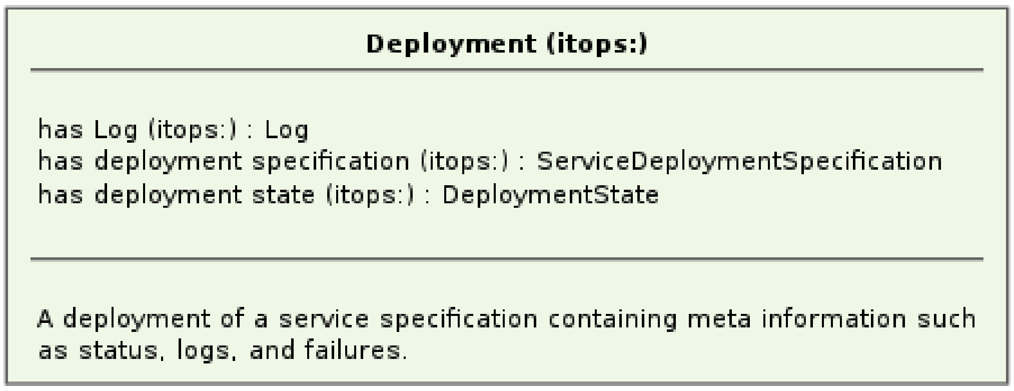

Create a new "Deployment"

Metadata

label *	A value is required	en (English)	
comment		en (English)	
state *	A value is required		
specification *	A value is required		

Figure 4.4 Deployment Creation Window

Deployment States When a user creates a Deployment it is marked with the state *Creating*. When the scheduler reaches awareness of the new Deployments, the Deployment moves to the state waiting and stays in this state until the full Deployment is deployed. After being deployed, the state is then moved to *Running* and stays there until the task is finished or stopped. A service may have two finish states *Finished Clean* or *Finished With Errors*. A service has a clean finish when it executes the task in a whole without exception and with errors otherwise. While waiting or after a service execution the user can check the Log metadata to check for execution errors, runtime, or service debugging. Following **Figure 4.5** and **Figure 4.6** gives an overview of either Deployment and Service Deployment Specification classes available in BRAINE vocabulary at https://github.com/eccenca/braine-vocab.

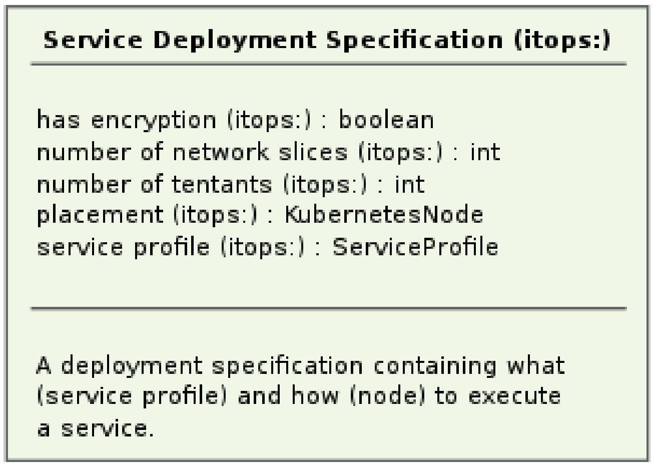


Deployment (itops:)

has Log (itops:) : Log
has deployment specification (itops:) : ServiceDeploymentSpecification
has deployment state (itops:) : DeploymentState

A deployment of a service specification containing meta information such as status, logs, and failures.

Figure **4.5** Deployment Class



Service Deployment Specification (itops:)

has encryption (itops:) : boolean
number of network slices (itops:) : int
number of tentants (itops:) : int
placement (itops:) : KubernetesNode
service profile (itops:) : ServiceProfile

A deployment specification containing what (service profile) and how (node) to execute a service.

Figure **4.6** Service Deployment Specification Class

Node Scheduling The architecture of each Node is already collected using the Kubernetes Node APIs, **Figure 4.7** shows the Node meta-information collected and managed by the BRAINE platform. It allows the user to identify the best Node for AI Service running, training, or testing by choosing the desirable Node architecture. With the collected running metadata, it is also possible to check possible running failures, data access and execution metadata by using the Log metadata from the Deployment class.

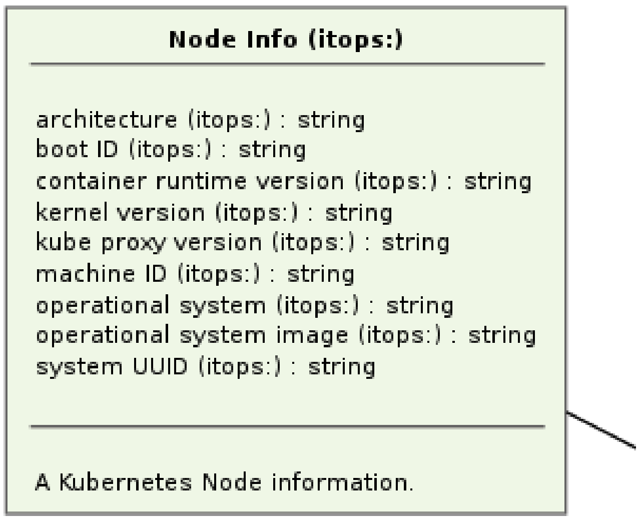
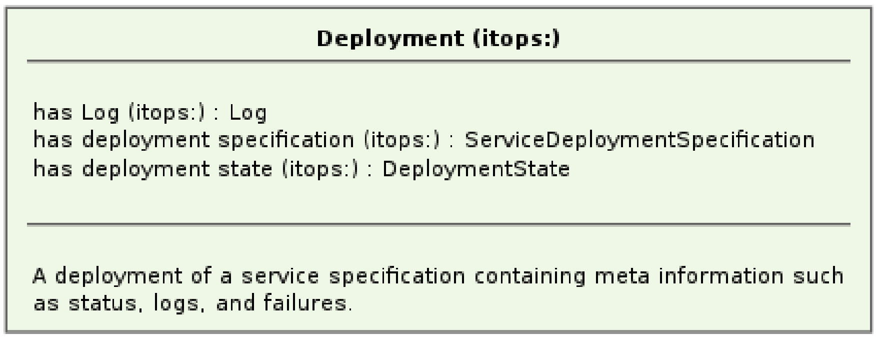

Node Info (itops:)
architecture (itops:) : string
boot ID (itops:) : string
container runtime version (itops:) : string
kernel version (itops:) : string
kube proxy version (itops:) : string
machine ID (itops:) : string
operational system (itops:) : string
operational system image (itops:) : string
system UUID (itops:) : string
A Kubernetes Node information.

Figure **4**.**7** Node Info metadata

Evaluating Service & Workflow Execution The service measurement and monitoring is performed through the classes Deployment and Log. The class Deployment contains metadata information about the Deployment state and the Log about the execution. Currently it is possible to register individual executions of every single agent involved in the Service. Thus, the user can check the system health and runtime through the Logs and promptly execute corrective actions in case of failures such as redeploying the Service in another Node. One feature that we want to evolve in the next version is to include the possibility of registering individual Log information as events such as Information, Error, and Debug. It also would be good to have a Log for each Service. As the possibility to create composed Services is still not possible, there is no necessity to have the Service specified. Another possibility is the addition of Data Access events in the log file such as Read and Write. To enable Memory and CPU monitoring by Deployment, there is a necessity to add Kubernetes PODs metadata to the Log Class as well the input parameters in the ServiceDeploymentSpecification.

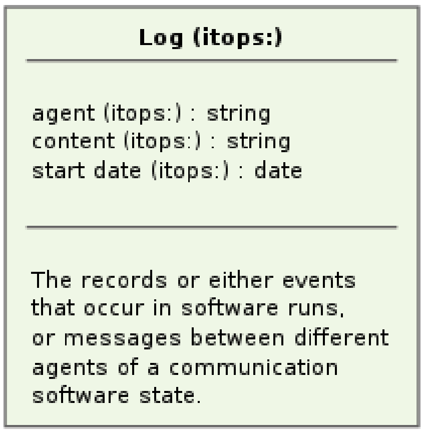
The same is valid for Networking, which we currently are not being registered. We intend to address these issues in the next Deliverable. Following **Figure 4.8** and **Figure 4.9** shows respectively Deployment and Log classes from BRAINE vocabulary at https://github.com/eccenca/braine-vocab.


Deployment (itops:)

has Log (itops:) : Log
has deployment specification (itops:) : ServiceDeploymentSpecification
has deployment state (itops:) : DeploymentState

A deployment of a service specification containing meta information such as status, logs, and failures.

Figure **4.8** Deployment class


Log (itops:)

agent (itops:) : string
content (itops:) : string
start date (itops:) : date

The records or either events that occur in software runs, or messages between different agents of a communication software state.

Figure **4.9** Log class

5. Measurements and monitoring

5.1. Flow based network telemetry framework

The telemetry framework is responsible to generate and collect telemetry information regarding the network node behaviour and the traffic passing over the network node.

It consists of the following sub-components:

- HW running P4 program, exporting telemetry and generating telemetry events
- P4 agent application that initialize and configure the HW P4 program to add/remove flows that should be monitored.
- Monitoring and export unit that processes raw data and converts it to a format that can be used by collectors
-

Following interfaces are used to communicate between components and external interfaces:

- P4 programs based on P4-lang to program data plane of network elements and telemetry collector
- gRPC to export the data from Network telemetry framework to adapter unit (see below Figure 5.1)

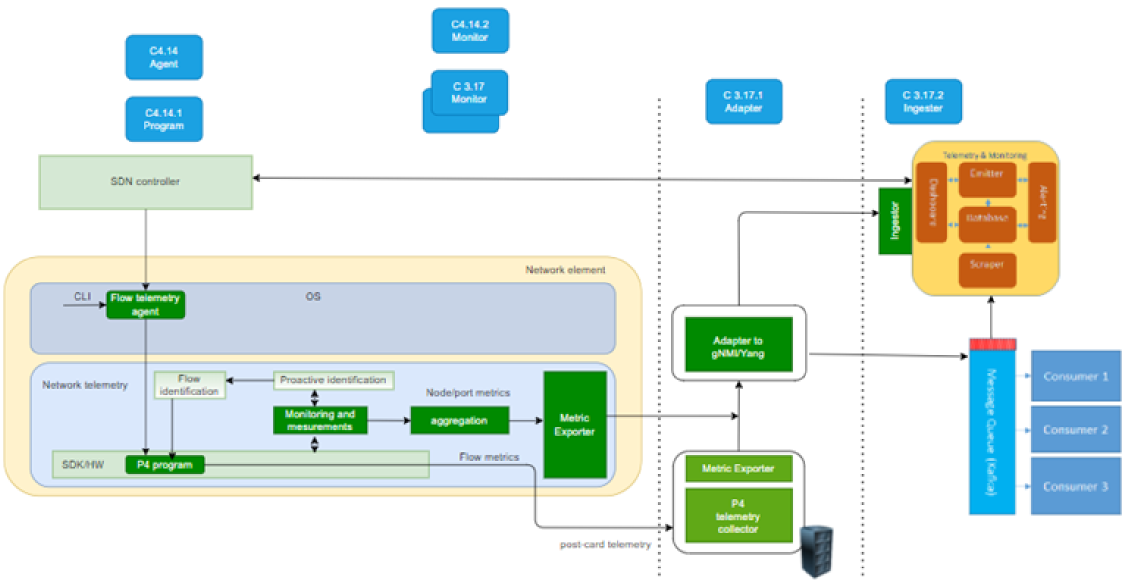


Figure 5.1 An overview of the Flow based network telemetry framework and components

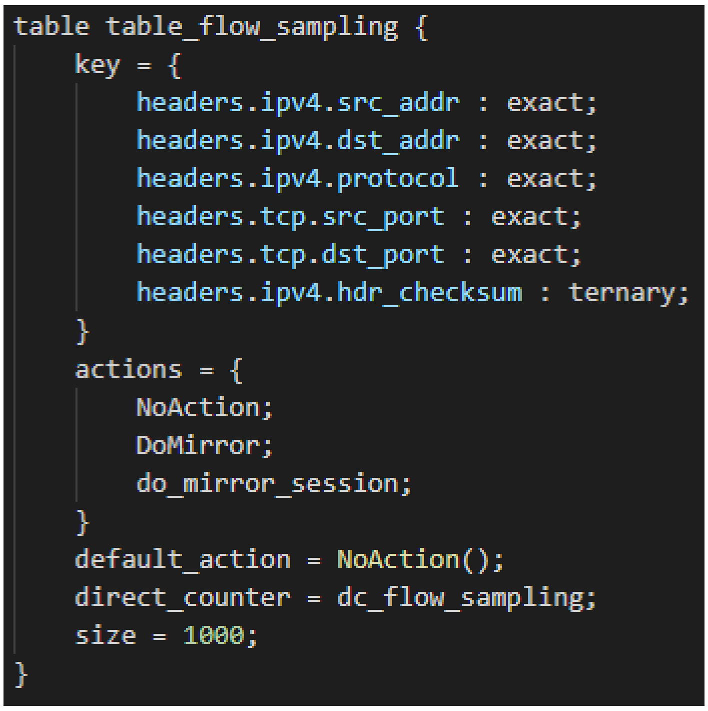
5.1.1. Flow telemetry Agent (C.4.14)

The Flow Telemetry agent is responsible to initialize and configure the P4 program that was auto-generated by MLNX P4 backend compiler, this agent's main user interface is a CLI running on SONIC. This agent also configures the needed HW capabilities to enable telemetry reporting (Mirror). An example configuration is a Flow (5 tuples) that should be monitored. Once the selected network session/flow was added to P4 tables, the HW will send telemetry events to the Telemetry monitor & exporter components.

5.1.2. Flow P4 program (C4.14.1)

The Flow telemetry P4 program is responsible to configure the low-level HW to support the P4 program written in P4-lang. In the flow telemetry case, this is a P4 table monitoring 5 tuples and mirroring the sampled traffic to the switch's CPU for reporting to the remote collector.

The P4 table (ACL) holds entries with 5 tuple keys and mirror actions, the Flow telemetry agent is responsible to add those entries. Below is the P4 table used for flow telemetry.



```
table table_flow_sampling {
    key = {
        headers.ipv4.src_addr : exact;
        headers.ipv4.dst_addr : exact;
        headers.ipv4.protocol : exact;
        headers.tcp.src_port : exact;
        headers.tcp.dst_port : exact;
        headers.ipv4.hdr_checksum : ternary;
    }
    actions = {
        NoAction;
        DoMirror;
        do_mirror_session;
    }
    default_action = NoAction();
    direct_counter = dc_flow_sampling;
    size = 1000;
}
```

Figure **5.2** P4 source code to define the flow telemetry table

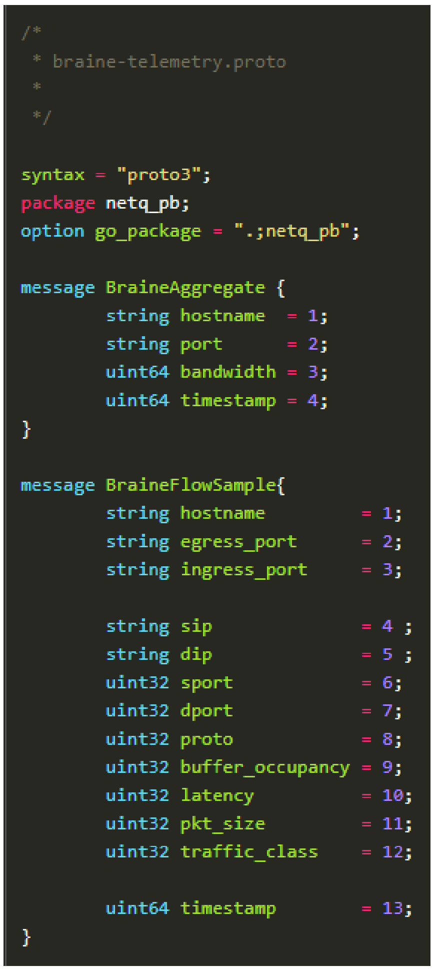
This program is auto-generated from the P4 source code by MLNX backend P4 compiler.

5.1.3. Telemetry Monitor and exporter (C4.14.2)

The telemetry monitor & exporter is responsible to collect and report telemetry data from network elements regarding network node behaviours and the traffic passing over the network node.

This component will wait for selected telemetry events from HW (that was configured by the P4 program C4.14.1) and will generate a report via gRPC to the Adapter component (C13.17.1).

Below is the example gRPC proto file that is used to stream data from the telemetry monitor & exported to the Adapter component



```
/*
 * braine-telemetry.proto
 *
 */

syntax = "proto3";
package netq_pb;
option go_package = ".;netq_pb";

message BraineAggregate {
        string hostname  = 1;
        string port      = 2;
        uint64 bandwidth = 3;
        uint64 timestamp = 4;
}

message BraineFlowSample{
        string hostname         = 1;
        string egress_port      = 2;
        string ingress_port     = 3;

        string sip              = 4 ;
        string dip              = 5 ;
        uint32 sport            = 6;
        uint32 dport            = 7;
        uint32 proto            = 8;
        uint32 buffer_occupancy = 9;
        uint32 latency          = 10;
        uint32 pkt_size         = 11;
        uint32 traffic_class    = 12;

        uint64 timestamp        = 13;
}
```

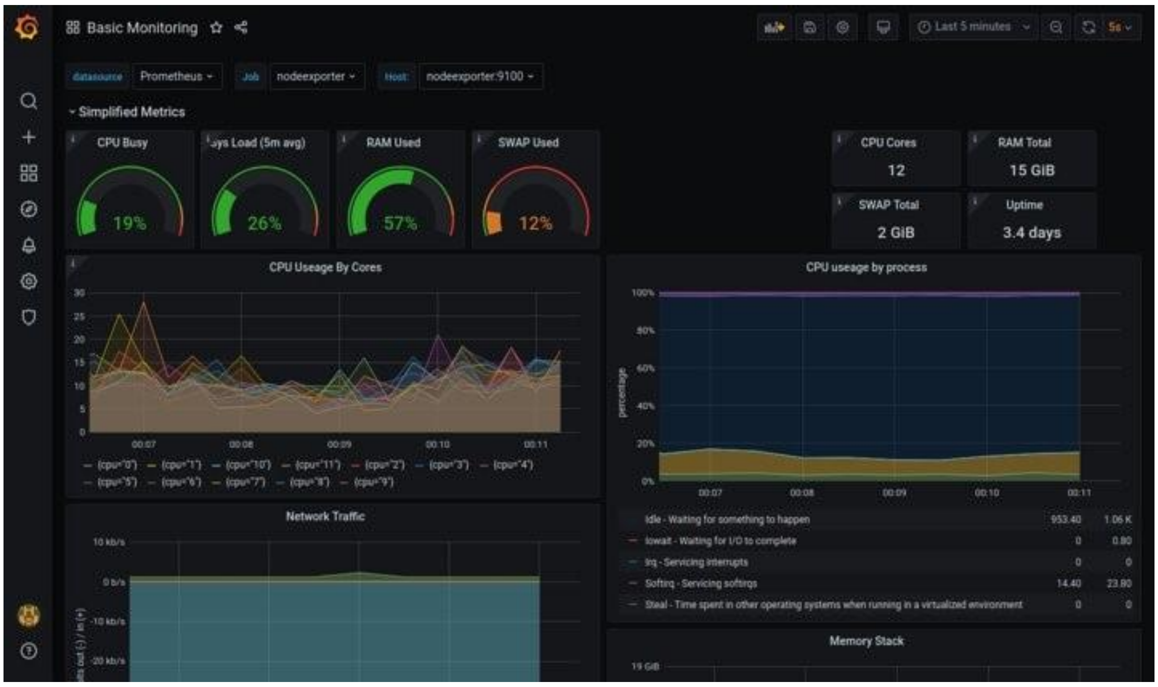
Figure **5.3** gRPC proto example used in The telemetry monitor & exporter

5.2. Monitoring and Dashboarding

Monitoring and dashboarding components are parts of the overall BRAINE telemetry system. The monitoring component processes events and generates alerts. The dashboarding component provides a central visual interface to disseminate information about the devices, the platform, and applications statuses.

At the moment, metric collectors and exporters send their metric data to a central InfluxDB instance. We use Grafana to obtain metric data from this database and visualize it via various dashboards and charts. It is able to query InfluxDB and Prometheus by using InfluxQL and PromQL, respectively to build custom datasets for visualization.

The dashboards for Grafana can be generated with a UI interface, or manually generated by creating/modifying a JSON file. We have also provided containerized visualization specification elements for quick and easy assembly of new customized Grafana dashboards. Each visualization specification can be manually added to a list, and when a corresponding python file is executed, it will develop a custom dashboard. There are requests to separate device and platform metrics from the application metrics and even further to separate either physically or logically the metric data of the applications. Also, there is a request for replacing the telemetry database for high availability and multi-cluster systems. Although these requests fall under WP3 here LUH is investigating their impact on monitoring and dashboarding.



Idle - Waiting for something to happen	953.40	1.06 K
Iowait - Waiting for I/O to complete	0	0.80
Irq - Servicing interrupts	0	0
Softirq - Servicing softirqs	14.40	23.80
Steal - Time spent in other operating systems when running in a virtualized environment	0	0

Figure 5.4 An example of the monitoring dashboard showing charts of CPU, Network Traffic, and Memory

6. Components

6.1. Data lifecycle manager (C4.1)

Component ID	Component Name	Development	Owner
C4.1	Data lifecycle manager	30%	DELL

GitLab Repository: In Progress

Containerized: Y

Registered on BRAINE platform image registry: N

Deployed as a pod and functional on BRAINE platform: N

Integrated with other platform components: Y – Apache Ozone global filesystem

Status Report:

Development of the data lifecycle manager is in progress, with current development and testing occurring with an internal test and integration with the global filesystem used within BRAINE. The manager is containerized and can connect to the Apache Ozone global filesystem, but development of a process to ingest filesystem events into an immutable ledger is ongoing.

6.2. Policy manager (C4.2)

Component ID	Component Name	Development	Owner
C4.2	Policy Manager	50%	DELL

GitLab Repository: In Progress

Containerized: Y

Registered on BRAINE platform image registry: N

Deployed as a pod and functional on BRAINE platform: N

Integrated with other platform components: In Progress

Status Report:

The policy manager is based on Apache Ranger, which integrates seamlessly with the global filesystem, Ozone. It must also integrate with the SLA Broker and lifecycle manager for effective data monitoring and enforcement of policies, which is ongoing.

6.3. Global File System (C4.3)

Component ID	Component Name	Development	Owner
C4.3	Distributed Data Storage	70%	DELL

GitLab Repository:

https://hub.docker.com/repository/docker/khalia6/ozone

https://hub.docker.com/repository/docker/khalia6/csi-node-driver-registrar

https://hub.docker.com/repository/docker/khalia6/csi-provisioner

Containerized: Y

Registered on BRAINE platform image registry: Y

Deployed as a pod and functional on BRAINE platform: Y

Integrated with other platform components:Y

Status Report:

The data storage is based on Apache Ozone and seamlessly stores data across heterogenous CPU architectures (ARM and AMD). The multi-tagged docker images have been prepared and tested. This component can be used by all the applications that require persistent storage. Efforts on interfacing and integrating the storage system with the data placement framework are ongoing.

6.4. Active Data Product (C4.4)

Component ID	Component Name	Development	Owner
C4.4	Active Data Product (ADP)	15%	LUH

GitLab Repository: https://github.com/braine-project/WP4R/tree/main/T41

Containerized: Y

Registered on BRAINE platform image registry: N

Deployed as a pod and functional on BRAINE platform: N

Integrated with other platform components: N

Status Report:

An active data product is indeed a dataset encapsulated in a secure container that allows access via a well-defined access point that conforms to the terms of an agreed-upon contract. A data product is said to be active as it can operate in an external environment. It is indeed a self-contained, secure executable package that must be run in order to allow for the utilization of the data it contains. When requested by external agents to access the data, it will ensure that the request and the response comply with contract terms, usage, sovereignty regulations, and boundaries defined.

A contract definition language using YAML is under development, by which the data owner can define the terms and conditions for accessing the data. The contract is then enforced by the contract controller. Any legitimate access to the data will be recorded in a blockchain for contract term enforcement as well as for auditing and accounting. The ADP component is a prototype and may not be secure enough for production environments with high-sensitive data.

Each UC that aims at sharing data with other UCs or external systems/parties may benefit from this component.

6.5. Catalyzr tool (C4.5)

Component ID	Component Name	Development	Owner
C4.5	Catalyzr Tool	50%	SIC

GitLab Repository:

Containerized: N

Registered on BRAINE platform image registry: N

Deployed as a pod and functional on BRAINE platform: N

Integrated with other platform components: integrated with the Global Service Registry.

Status Report: Catalyzr is a tool for joining work between cryptographer and software developer for hunting security vulnerability. The capability to track microarchitecture-induced vulnerabilities (e.g., Spectre / Meltdown breach class) is operational. The Catalyzr is now deployed for join work between SIC and ISW to secure ISW cryptographic software libraries.

6.6. Authoring tool (C4.6)

Component ID	Component Name	Development	Owner
C4.6	Authoring Tool	50%	ECC

GitLab Repository: https://github.com/eccenca/braine/tree/main/webclient

Containerized: N

Registered on BRAINE platform image registry: N

Deployed as a pod and functional on BRAINE platform: N

Integrated with other platform components: N – The Authoring Tool haven't been integrated with the Global Service Registry.

Status Report:

The Authoring Tool for service composition is under development. The data model to support persistence through the Resource & Service Catalog is already implemented while the development of the user interface has been already initiated. In the next iterations we expect to have a functional and integrated version working.

6.7. Service Orchestrator (C4.7)

Component ID	Component Name	Development	Owner
C4.7	Service Orchestrator	50%	ECC

GitLab Repository: https://github.com/eccenca/braine/tree/main/service-orchestrator

Containerized: N

Registered on BRAINE platform image registry: N

Deployed as a pod and functional on BRAINE platform: N

Integrated with other platform components: Y – The Service Orchestrator synchronize service metadata such as status between the Global Service Registry and the Resource & Service Catalog.

Status Report:

The Service Orchestrator is under development being partially functional and integrated, however it needs further testing and development.

6.8. Monitoring Dashboard (C4.8)

Component ID	Component Name	Development	Owner
C4.8	Monitoring Dashboard	95%	LUH
GitLab Repository: https://gitlab.com/braine/wp4-monitoringsystem-luh Containerized: Y Registered on BRAINE platform image registry: Y Deployed as a pod and functional on BRAINE platform: Y Integrated with other platform components: Y Status Report: The monitoring dashboard is a visualization system for the time-series metric data that are stored in InfluxDB (and Prometheus). The dashboard comes with a pre-configured set of gauges and charts that display various system metrics scraped from node-exporter, including CPU, memory, disk I/O writing, and network traffic metrics. It is also able to visualize additional time-series data generated by the UC applications. The monitoring dashboard relies on the telemetry infrastructure components such as scraper and database from WP3. Each device (network switch, compute node), platform component (operating system, scheduler, data lifecycle manager, etc), and use case may generate metrics and send them to the telemetry database for storage and processing. The monitoring dashboard can be tuned to extract general or specific (use case-related) metric data, filter and aggregate them and then display charts, gauges, or other visual forms of the data.			

6.9. Data Placement (C4.9)

Component ID	Component Name	Development	Owner
C4.9	Data Placement Framework	85%	UCC
GitLab Repository: https://gitlab.com/braine/c49dataplacementframework Containerized: Y Registered on BRAINE platform image registry: N			

Deployed as a pod and functional on BRAINE platform: N Integrated with other platform components: N Status Report: The first draft of data placement framework is almost completed, just improving it further for overall better integration with the platform. Each UC may have its own types of constraints on the data, the placement framework stores and uses these constraints during selecting the data nodes for the data store requests generated by the UC. The data placement framework works with modified Ozone. This is an external data placement framework, built as an API in Java. This API uses the available information for data placement. It keeps the record of the present state of data allocation on various data nodes. The API also has the details of the constraints on the data defined by the application. These constraints may be on node sharing, hardware (Intel, AMD, ARM), selected data nodes, or some specific location, etc. All these details are stored in config (XML and properties) files. The modified Ozone requests this API to get the data node list to store the input data. Based on the available information about this new data request (application, user, persistent volume, etc.) and the stored information about the current allocation the data placement API returns the list of data nodes to Ozone. Finally, Ozone uses these nodes to store the data.

6.10. Healthcare Assisted Living (C4.10)

AI-driven Digital Twin solution for new digital ecosystems enabling Smart Healthcare in Medical and Caregiving Centres Healthcare

Component ID	Component Name	Use Cases	Owner
C4.10	Exporter for the metrics for the UC1 application ‘AI-driven Digital Twin solution for new digital ecosystems enabling Smart Healthcare in Medical and Caregiving Centres’	UC1	IMC
GitLab Repository: private repository Containerized: Y Registered on BRAINE platform image registry: N Deployed as a pod and functional on BRAINE platform: Y			

Integrated with other platform components: work in progress

Status Report:

The use ‘Healthcare Assisted Living: AI-driven Digital Twin solution for new digital ecosystems enabling Smart Healthcare in Medical and Caregiving Centres’. The goal of the application to create a digital twin of patients using microservices and continuous collection and analysis of patient data. As part of the task of ‘WP4: User-oriented utilization of the edge’ additional component was designed and developed as part of the adaptation the Edge-based system for human-centric applications.

Key metrics required for the UC1 monitoring were defined and relevant for the UC1 application, monitoring tool— metric log connector in short ‘exporter’—was designed and developed for the telemetry and application monitoring. The exporter for the metrics for the UC1 application connects to the C3.6 and provides an endpoint "/metrics" and sends GET metrics on request from the Prometheus server. A custom for UC1 application exporter is deployed as a pod and is functional on BRAINE platform.

The exporter written in Go language, deployed as pod in the system and added to the service which will be accessed by Prometheus to provide the set of metrics (as required by Prometheus' exporter implementation).

The overall view of metrics for the UC1 application which runs on EMDC is as follows (the number of actual metrics is bigger):

- Queue size for command execution, by using the label "queue_type" we make a separation into several queue types. In doing so we can use one metric and several labels to get a time series for all queues in the system;
- Command execution time in seconds;
- The number of imported values per execution;
- Number of indicator values calculated per command call;
- Number of object state values calculated in one call to the object state calculation command;
- Number of active generators;
- Total number of registered users in the system;
- Total number of models in the system;
- Total number of sensors;
- Total number of data elements;
- Total number of indicators;

- Total number of indicator values for all indicators;
- Total number of sensor values for all sensors;
- The total number of values of the data items for all data elements.

6.11. Motif Discovery Tool (C4.11)

Component ID	Component Name	Development	Owner
C4.11	Motif Discovery Tool	85%	FS

GitLab Repository: https://gitlab.com/braine/wp4-mod-discovery-module-fs

Containerized: Y

Registered on BRAINE platform image registry: Y

Deployed as a pod and functional on BRAINE platform: Y

Integrated with other platform components: Y

Status Report:

The Motif Discovery Module of Motif Discovery Tool (MOD) enables the discovery of all repetitive patterns of any size in any time-series data. The time-series data from the perspective of BRAINE are sensory data from machines and devices on the shop floor. The discovered patterns represent unique operations of the machine.

MOD is divided into several containerized modules, which can be deployed individually on different host machines. Within WP4, the Discovery module and the Detection module are being developed and implemented.

Components were tested on CNIT Braine Testbed. Currently, these components are being integrated within the UC3 and Learning Module of Motif Discovery Tool (WP3).

6.12. vRAN with adjustments (C4.12)

Component ID	Component Name	Development	Owner
C4.12	vRAN with adjustments	50%	ISW

GitLab Repository: Link

Containerized: **N**

Registered on BRAINE platform image registry: **N**

Deployed as a pod and functional on BRAINE platform: **N**

Integrated with other platform components: **N** – not yet. Only when the model testing (prototype implementation) will be positively validated.

Status Report:

Here the PDCP acceleration with FPGA has been validated – but the results were not satisfactory for ISW due to increase in latencies on the path CPU-FPGA for the PDCP PDUs sent to the encryption engine in FPGA. The next steps are undergoing internal discussions at the stage of D4.2 issue date and will be reported in the D4.3 deliverable.

6.13. AI platform profiling engine (C4.13)

Component ID	Component Name	Development	Owner
C4.13	AI platform profiling engine	70%	NEC

GitLab Repository: **N**

Containerized: Y

Registered on BRAINE platform image registry: **N**

Deployed as a pod and functional on BRAINE platform: **N**

Integrated with other platform components: Y – UC2 video analysis application integrated

Status Report:

The overall tool is integrated into applications by leveraging direct import in high-level domain-specific machine learning frameworks, such as PyTorch and TensorFlow, and therefore it can be shipped jointly with the application deployment units.

6.14. Network Telemetry Framework (C4.14)

Component ID	Component Name	Development	Owner
C4.14	Flow telemetry agent	70%	MLNX

GitLab Repository: N

Containerized: Y

Registered on BRAINE platform image registry: N

Deployed as a pod and functional on BRAINE platform: N

Integrated with other platform components: Y – Info: Integrated with Flow P4 program (C4.14.1)

Status Report:

- The flow telemetry agent was tested to add/remove selected traffic flow, the P4 tables are updated with the add/remove entries and telemetry events are sent to Monitor & exported (C4.14.2).
- Event rate is still work in progress.

Component ID	**Component Name**	**Development**	**Owner**
C4.14.1	Flow P4 Program	90%	MLNX

GitLab Repository: N

Containerized: Y

Registered on BRAINE platform image registry: N

Deployed as a pod and functional on BRAINE platform: N

Integrated with other platform components: Y – Info: Integrated with Flow telemetry monitoring and exporter (C4.14.2) for streaming telemetry data

Status Report:

The P4 program code is done, HW tables are created and new flows can be added to those tables.

Component ID	**Component Name**	**Development**	**Owner**
C4.14.2	Flow telemetry monitor & exporter	70%	MLNX

GitLab Repository: N

Containerized: Y

Registered on BRAINE platform image registry: N

Deployed as a pod and functional on BRAINE platform: N

Integrated with other platform components: Y – Info: Integrated with Telemetry Adapter for streaming telemetry data.

Status Report:

HW telemetry events are collected by the component and also exported to remote collector via gRPC.

7. Conclusions

This document provides the status of BRAINE WP4 activities at end of Year 2.

This deliverable first reports on implementation of the data lifecycle management and the distributed storage system. Then, it provides the specification and realization of a data placement framework that is able to place data on nodes based on privacy and security constraints. Futhermore, it reports on the AI application ensuring optimal hardware utilization. This deliverable also provides the workflow definition language based on OWL-S for modelling atomic, simple, and composite processes, successfully integrated with the Kubernetes-compatible open-source workflow execution framework Argo. Finally, this document reports on the P4 based programs for collecting and exporting telemetry records from the switches.