D6.1 FRACTAL processing node design and implementation

Deliverable Id:	**D6.1**
Deliverable name:	***FRACTAL processing node design and implementation***
Status:	**Ready**
Dissemination level:	**Public**
Due date of deliverable:	**2022-28-02(M18)**
Actual submission date:	**2022-16-03**
Work package:	**WP6 CPS Communication Framework**
Organization name of lead contractor for this deliverable:	**HALTIAN**
Authors:	Matti Vakkuri, HALTIAN Jyrki Okkonen, HALTIAN Antti Takaluoma, OFFCODE Susanna Pirttikangas, UOULU Vahid Mohsseni, UOULU Nanna Setämaa, UOULU Huong Nguyen, UOULU Mickaël Bettinelli, UOULU
Reviewers:	Ester Sola, ZYLK, esola@zylk.net Roman Obermaisser, SIEG, roman.obermaisser@uni-siegen.de

Abstract:

This deliverable provides the software design and implementation of the edge processing node, including a fractal configuration and features to enable scalability in the platform.

This project has received funding from the ECSEL Joint Undertaking (JU) under grant agreement No 877056

Co-funded by the Horizon 2020 Programme of the European Union under grant agreement No 877056.

Contents

1 History

Version	Comment	Contributors	Reviewers
1.0	Version to reviewers	Matti Vakkuri,	Ester Sola, Roman Obermaisser
1.1	Iteration after the first reviewer comments	Matti Vakkuri, Vahid Mohsseni	Ester Sola, Roman Obermaisser
1.2	Iteration after the second reviewer comments	Matti Vakkuri, Vahid Mohsseni	Ester Sola, Roman Obermaisser
1.3	Pre-Final version	Matti Vakkuri, Vahid Mohsseni	Ester Sola, Roman Obermaisser
1.4	Final version	Matti Vakkuri, Vahid Mohsseni	Ester Sola, Roman Obermaisser

Table 1 – List of document versions

2 Summary

The scope of the deliverable D6.1 FRACTAL processing node design and implementation is to in practice design the node architecture and implement it.

D6.1 is directly related to task T6.1 Edge node design and implementation.

In other terms, the purpose is to develop and deploy the necessary edge computing infrastructure. For this purpose, a preliminary analysis of existing open-source edge computing-based software platforms Apache Kura, EdgeX Foundry, StarlingX, OpenEdge Kubernetes and Microsoft Azure IoT Edge, was done to select a reference implementation.

As a set of requirements from FRACTAL Deliverable D2.1 an open-source software implementation has been obtained.

Initial methodological framework specification introduced in D2.1 gives guidelines for building processing node design and its implementation with guiding principles:

- Keep the architecture as modular as possible and, for this reason
- Containers and microservice technologies to be studied as possible approaches
- Predefining tools for some of the data lifecycle phases.

In D6.1 the required set of software components and tools useful for the FRACTAL engineering framework were selected, updated, and implemented. These components enable core functionalities such as e.g., application provisioning and scheduling, application containerization and deployment or dynamic balancing of workload to the available infrastructure resources.

3 Introduction

This deliverable D6.1 provides the **software design and implementation** of the edge processing node, including a fractal configuration and features to enable scalability in the platform. As a result, an open-source software implementation is obtained including core functionalities, core microservices for common operations within the FRACTAL engineering framework and appropriate mechanisms to support remote monitoring, resource management and dynamic reconfiguration of the edge nodes functionalities for remote monitoring and management of the FRACTAL nodes.

This deliverable is directly related to task 6.1 in WP6.

The deliverable concentrates on the area of a processing platform at the edge with connection definition to different IoT devices and cloud platforms. All other aspects have been or are to be delivered in other WPs and tasks, and therefore are connected to, but out of scope of this deliverable.

3.1 High level architectures

For background and for the justification of the practical architectural work, the analysis of four different high-level architectures took place to align the design with several existing edge architectures. [1]

- Reference architecture Model Edge Computing (RAMEC)
- Smart Grid Architecture Model (SGAM)
- Reference architecture model for *Industrie 4.0 (RAMI4.0)*
- Industrial Internet Consortium (IIRA).

3.1.1 Industrial Internet Consortium (IIRA)

The IIRA[2] is a standards-based open architecture for IoT systems. The IIRA maximizes its value by having broad industry applicability to drive interoperability, to map applicable technologies, and to guide technology and standard development. The architecture description and representation are generic and at a high level of abstraction to support the requisite broad industry applicability.

1 A. Willner and V. Gowtham, "Toward a Reference Architecture Model for Industrial Edge Computing," in *IEEE Communications Standards Magazine*, vol. 4, no. 4, pp. 42-48, December 2020, doi: 10.1109/MCOMSTD.001.2000007.
2 https://www.iiconsortium.org/pdf/IIRA-v1.9.pdf

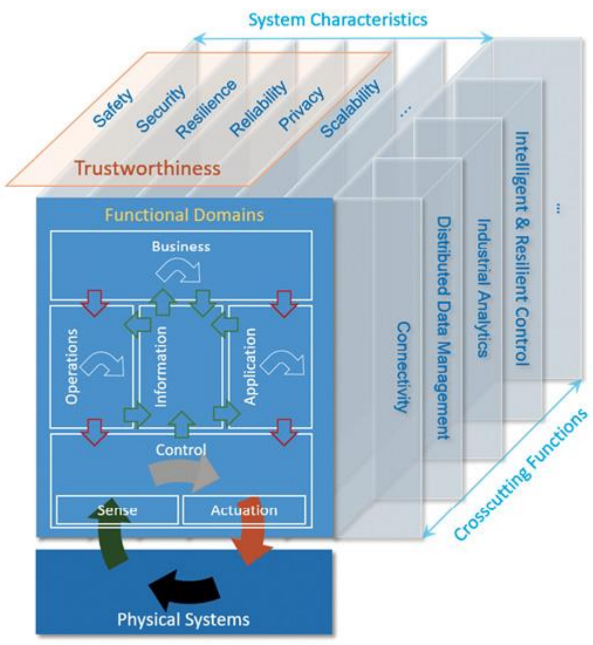
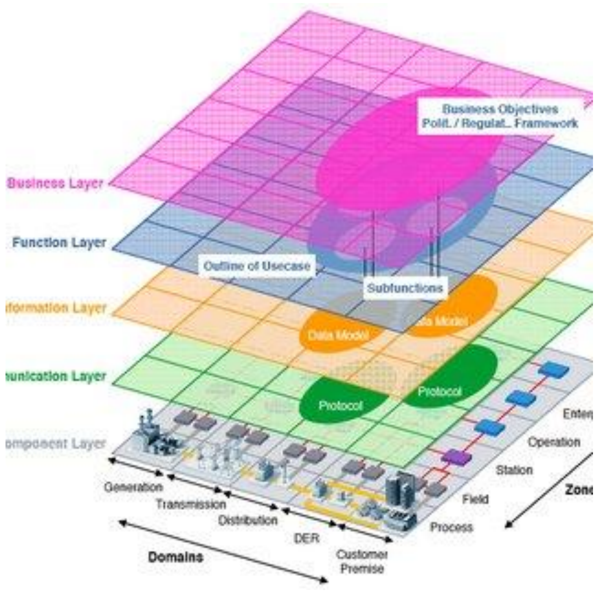


Figure 1 – Industrial Internet Consortium (IIRA) architecture

3.1.2 Smart Grid Architecture Model (SGAM)

The Smart energy Grid Architecture Model (SGAM)[3] is a three-dimensional architectural framework that can be used to model interactions (mostly exchange of information) between different entities located within the smart energy area.



3 https://ec.europa.eu/energy/sites/ener/files/documents/xpert_group1_reference_architecture.pdf

Figure 2– Smart energy Grid Architecture Model (SGAM)

3.1.3 Reference architecture Model Edge Computing (RAMEC)

Reference Architecture Model Edge Computing (RAMEC)[4] identifies 210 views on the Edge Computing paradigm in the manufacturing domain. Future iterations of this model might be used for the classification of relevant research, standardization, and development activities.

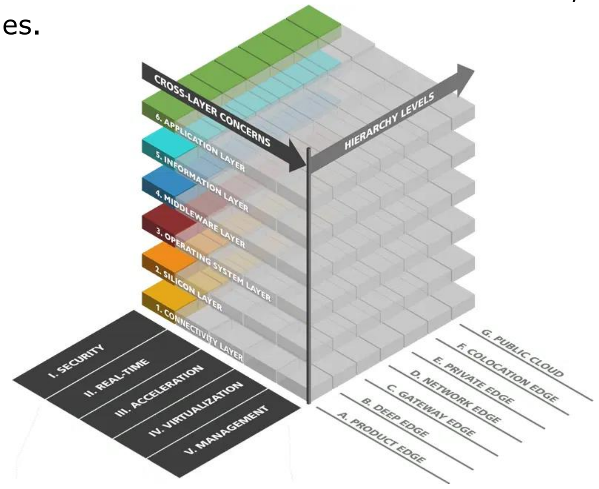
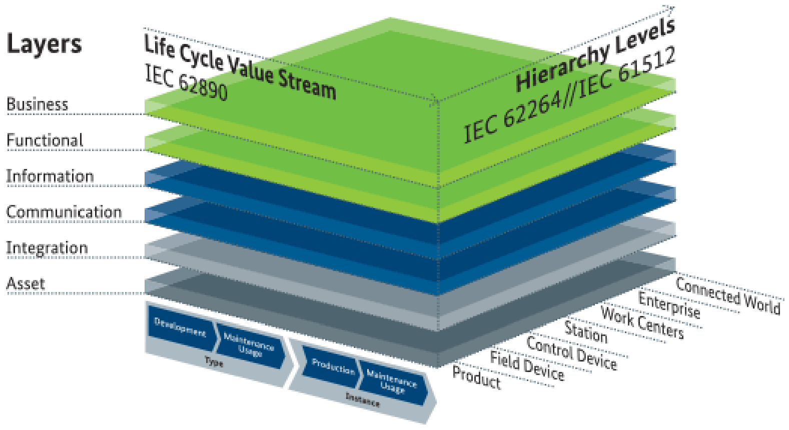


Figure 3– Smart energy Grid Architecture Model (SGAM)

3.1.4 Reference architecture model for Industrie 4.0 (RAMI4.0)

RAMI 4.0 Reference Architectural Model and the Industry 4.0[5] components give companies a framework for developing future products and business models. RAMI 4.0 is a three-dimensional map showing how to approach the deployment of Industry 4.0 in a structured manner. A major goal of RAMI 4.0 is to make sure that all participants involved in Industry 4.0 discussions and activities have a common framework to understand each other. The RAMI 4.0 framework is intended to enable standards to be identified to determine whether there is any need for additions and amendments. This model is complemented by the Industry 4.0 components.



4 https://ecconsortium.eu/
5 https://ec.europa.eu/futurium/en/system/files/ged/a2-schweichhart-reference_architectural_model_industrie_4.0_rami_4.0.pdf

Figure 4– Reference architecture model for Industrie 4.0 (RAMI4.0)

3.1.5 Conclusion of high-level architectures

These different architecture models have acceleration, API, communication, connectivity, data model, orchestration, integration, monitoring, privacy, resilience, reliability, real-time, scalability and security as common nominators as a set of requirements for more practical architecture considerations.

Common	**IIRA**	**SGAM**	**RAMEC**	**RAMI 4.0**
Acceleration	Acceleration		Acceleration	
API	API	API	API	API
Communication	Communication			Communication
Connectivity	Connectivity	Connectivity		
Data model	Data model	Data model		Data model
Orchestration	Deployment	Resource identification	Management	Life-cycle
Integration		Integration		Integration
Monitoring	Monitoring	Logging		
Privacy	Privacy	Privacy	Privacy	
Resilience	Resilience			Resilience
Reliability	Reliability	Reliability		
Real-time			Real-time	Real-tine
	Safety			
Scalability	Scalability	Scalability		
Security	Security	Security	Security	Security

Table 2 – Common nominators

3.2 Open-source edge computing-based software platforms

5 different edge platforms were under analysis with a target to select a reference implementation, if possible. After analysis Kubernetes with Docker was chosen to be the candidate for FRACTAL reference node implementation.

Compared platforms:

- EdgeXfoudry
- Kura
- Kubernetes
- Starlinx
- Azure IoT

3.2.1 EdgeXfoundry

EdgeXfoudry is highly flexible open-source software framework that facilitates interoperability between heterogeneous devices and applications at the IoT Edge, along with a consistent foundation for security and manageability. The open, vendor-neutral platform speeds developer and technology providers time to market by providing modular reference services for device-data ingestion, normalization, analysis and sharing in support of new IoT data services, advanced edge computing applications, including AI and automation.[6]

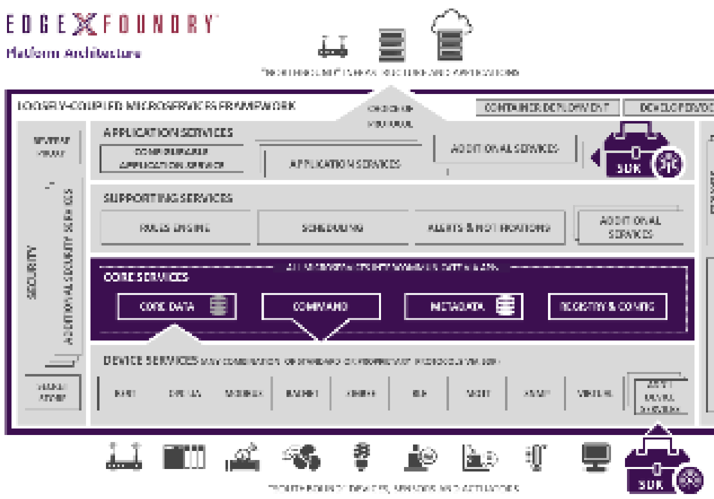


Figure 5– EdgeXfoudry architecture

3.2.2 Kura

Eclipse Kura™ is an extensible open source IoT Edge Framework based on Java/OSGi. Kura offers API access to the hardware interfaces of IoT Gateways (serial ports, GPS, watchdog, GPIOs, I2C, etc.). It features ready-to-use field protocols (including

6 https://www.edgexfoundry.org/

Modbus, OPC-UA, S7), an application container, and a web-based visual data flow programming to acquire data from the field, process it at the edge, and publish it to leading IoT Cloud Platforms through MQTT connectivity.[7]

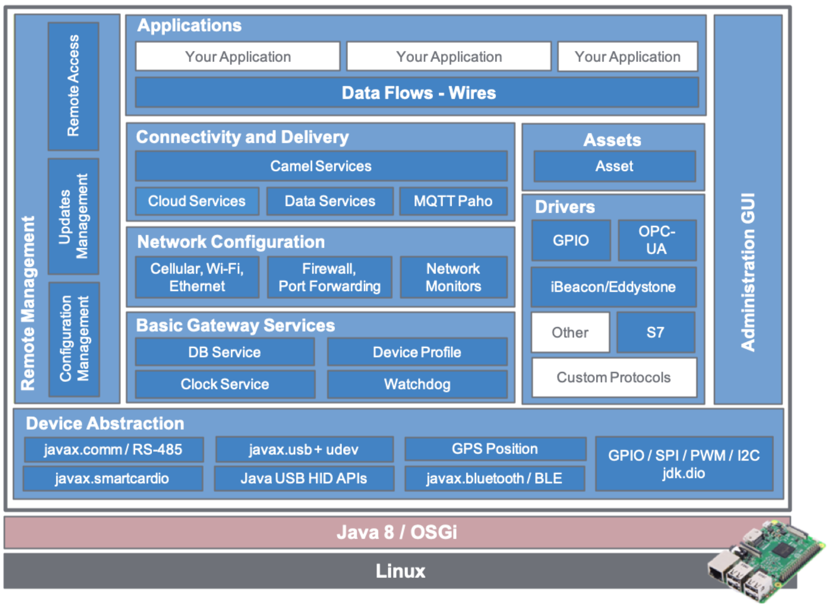


Figure 6– Eclipse Kura IoT architecture

3.2.3 Kubernetes

Kubernetes, also known as K8s, is an open-source system for automating deployment, scaling, and management of containerized applications. Kubernetes is highly configurable and extensible.[8]

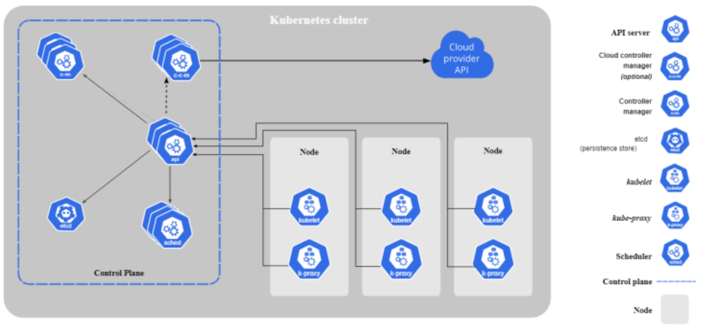


Figure 7– Kubernetes architecture

7 https://www.eclipse.org/kura/
8 https://kubernetes.io/

3.2.4 Starlingx

StarlingX is a complete cloud infrastructure software stack for the edge used by the most demanding applications in industrial IOT, telecom, video delivery and other ultra-low latency use cases.[9]

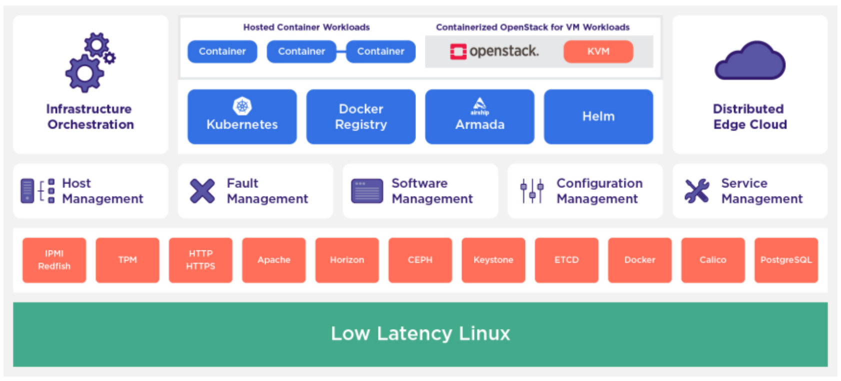


Figure 8– StarlingX architecture

3.2.5 Azure IoT

The Azure Internet of Things (IoT) is a collection of Microsoft-managed cloud services that connect, monitor, and control billions of IoT assets. In simpler terms, an IoT solution is made up of one or more IoT devices that communicate with one or more back-end services hosted in the cloud.[10]

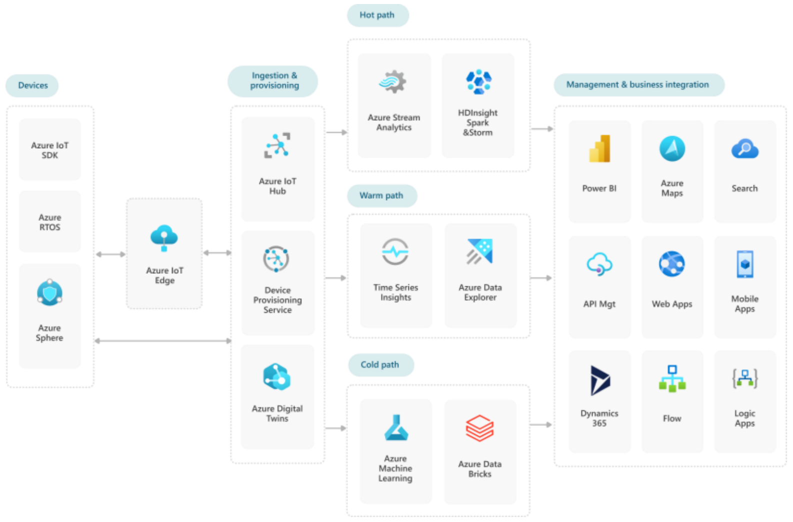


Figure 9– Azure IoT architecture

9 https://www.starlingx.io/
10 https://azure.microsoft.com/en-us/overview/iot/

3.2.6 Conclusion

The comparison gave a clear view that Kubernetes family suites most the approach of interoperability and integrations with other systems. Also, it is a preferred orchestration platform for Ubuntu operating system and works seamlessly with container platform Docker. Kubernetes is open to extensions, which provides more openness and fits into fractality design principles.

Implementation explained in Chapter 6 is based on Kubernetes family Microk8S and k3S as it brings lightweight, fully-featured, conformant Kubernetes for IoT devices.

4 Background

Artificial Intelligence is key for the IoT to enhance existing services and to operate in a more efficient manner. If AI is not implemented in the IoT its scope is very much limited. Cognitivity, provided by Artificial Intelligence, will support the IoT to adapt to surrounding world changes, learning in real-time to improve its performance.

The goal of FRACTAL project is to create a basic building block called the FRACTAL node. This building block is a reliable computing platform node able to build a Cognitive Edge (a network that makes predictions and diagnoses) under industry standards. The FRACTAL node will be the building block of scalable decentralized Internet of Things (ranging from Smart Low-Energy Computing Systems to High-Performance Computing Edge Nodes).

The strategic objectives of FRACTAL node are:

- O1: **Design and Implement an Open-Safe-Reliable Platform** to Build Cognitive Nodes of Variable Complexity
- O2: Guarantee FRACTAL nodes and systems extra-functional properties (**dependability, security, timeliness, and energy-efficiency**)
- O3: Evaluate and validate the analytics approach by means of AI to help the identification of the largest set of working conditions still preserving safety and security operational behaviors.
- O4: To **integrate fractal communication properties** (scale free networks) to FRACTAL nodes.

5 Architecture

5.1 Architecture design

The main design principle of Fractal node processing architecture has been openness and implement all possible as a microservice and as a container.

"A container is a standard unit of software that packages up code and all its dependencies, so the application runs quickly and reliably from one computing environment to another."[11]

FRACTAL architecture is hierarchical, meaning that upper layers are built upon, and consumed data and services provided by, lower layers. It is noticeable that sole the layers are overlapping and clear division between the roles is some level undefined.

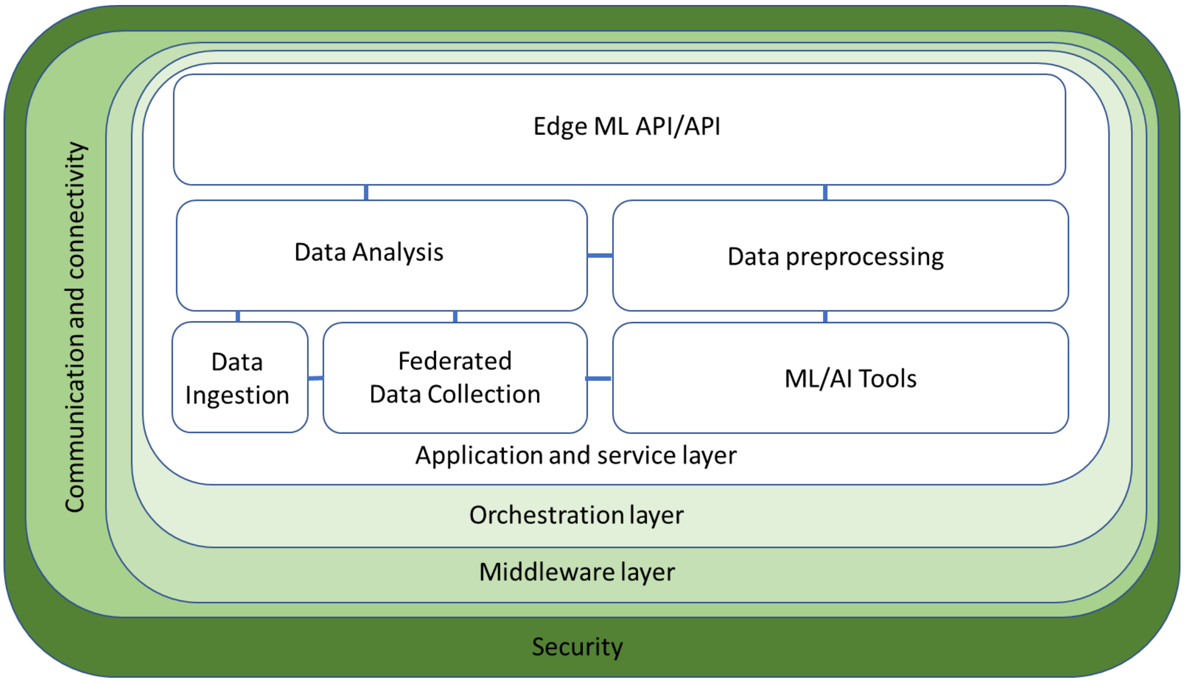


Figure 10 – FRACTAL Edge node processing architecture

11 https://www.docker.com/resources/what-container

5.2 Application and service layer

The Application layer contains modules implement the core functions for visualization, Control, Analytics. Data fusion, Filtering, storage database.

The idea in the architecture design and its application layer implementation that containers are used as much as applicable. Container a processing entity standardized from having dependencies to achieve same behavior wherever is run, on the edge or the cloud.

Containers decouple applications from underlying host infrastructure. This makes deployment easier in different cloud or OS environments.

5.3 IoT Middleware layer

The IoT middleware layer includes APIs, libraries, and SDKs, event logging, reporting tools, and user interfaces for interacting with the IoT deployment, its services, and its status.

IoT middleware is software that serves as an interface between components of the IoT, making communication possible among elements that would not otherwise be capable. Middleware is part of the architecture enabling connectivity for huge numbers of diverse Things by providing a connectivity layer for sensors and also for the application layers that provide services that ensure effective communications among software.[12]

5.4 Communication and connectivity layer

Communication and connectivity layer provides multi-protocol data communication between devices at the edge, devices, and gateways[13]. All the intermediate elements required in terms of hardware and software to exchange data between the data center and the network devices are handled on this layer.

5.5 Device layer

Device layer is the end-devices is hardware edge-nodes. In this layer, devices have a computing unit running either a single main programs or a real-time operating system (RTOS) that can use system calls or drivers to interact with low-level peripherals, interfaces, and attached hardware.

12 https://internetofthingsagenda.techtarget.com/definition/IoT-middleware-Internet-of-Things-middleware
13 https://www.nabto.com/guide-iot-protocols-standards/

5.6 Security layer

Security layer Secures all layer's modules within the data integrity and security including security policies, di0rectory management, authorization including certificates and tokens, authentication, encryption, integrity data checks.

5.7 Orchestration layer

IoT device, middleware, security, application, and usage orchestration as the process of integrating IoT applications with enterprise information technology systems, cloud services, mobile applications within and across the boundaries including log processing and forwarding as part for the whole life-cycle management.

5.8 Generic data model

The purpose of a generic data model to it as a building block for FRACTAL node.

The idea is that the data model's structure is easily readable by humans and machines, is compact, does not include unnecessary data notations with straightforward syntax and have a large support in modern programming languages and application platforms.

6 Implementation

6.1 Introduction

The implementation of a FRACTAL processing architecture is an example of an approach how the architecture fits in to the need and can be executed by using open-source applications.

In order to implement processing layer, the middleware layer needs to be installed (Operating system and its extensions)

For initial implementation Ubuntu was selected as primary operating system to demonstrate the whole stack implementation. For the basis, Ubuntu is the reference platform for Kubernetes on all major public clouds. In FRACTAL two different hardware architectures have been selected to run Linux ARM64 and 64-bit RISC-V. Note that low-end implementation is based on NuttX operating system with 32-bit RISC-V.

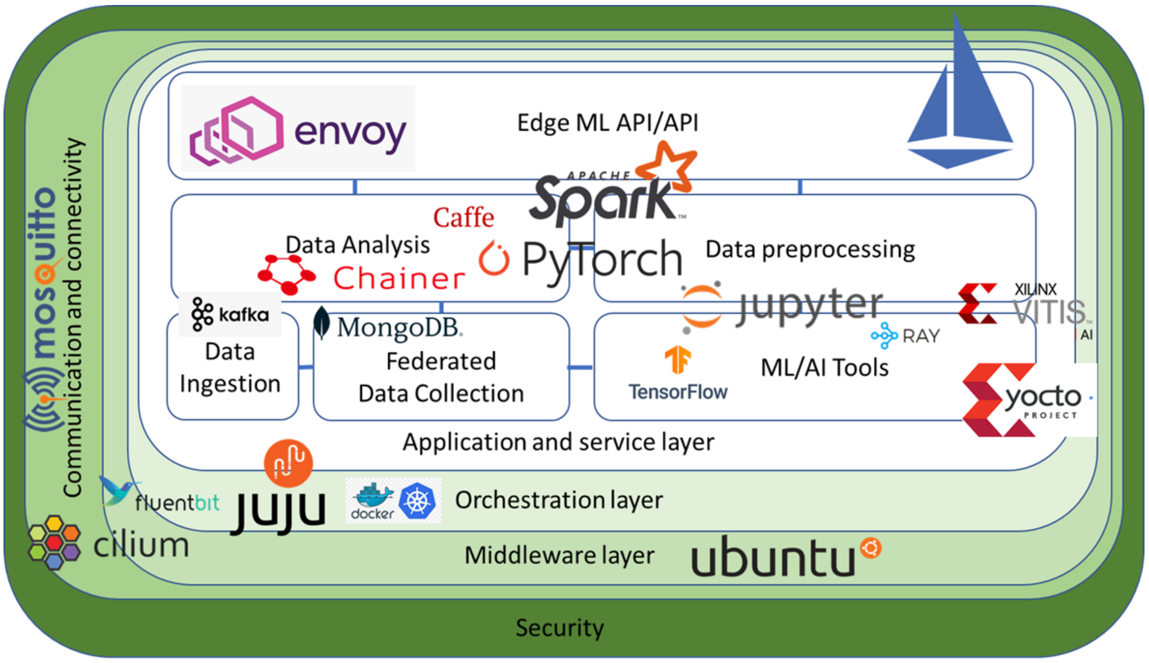


Figure 11 – FRACTAL Edge node processing architecture implementation

Note, the implementation is **an example of** how the architecture design can be implemented in practice to the real environment. There are several different possibilities to implement the set of layers above the hardware. In application layer the implementation example has been demonstrated the way that Docker images are available via internet from Docker Hub and alternatively to be installed locally as microservices from files. Microservices can be manually built e.g in the cloud and when cluster orchestration is available to deploy needed microservices through distributed orchestration. Also, software can be installed locally as described in its installation guide.

6.2 High-end node (ARM64)

High End node implementation example is based on VERSAL VKC-190 ARM64 architecture.

6.2.1 Middleware layer

The middleware layer is mostly part if WP3 and described in D3.2 Preliminary Fractal Software node and services, but to enable the implementation of the architecture the middleware installation is required.

Also, Ubuntu offers ready packaged Software/orchestration layer tools such as Microk8s, Docker, Mosquitto, Prometheus and Juju to be installed simultaneously with the operating system installation.

Operating system

The needed normal Ubuntu server install for ARM64.

https://cdimage.ubuntu.com/releases/20.04/release/ubuntu-20.04.3-live-server-arm64.iso

6.2.1.2 ARM-64 Emulation

Simulated environment for implementation simulation has been QEMU. ARM64 QEMU setup is following:

Preparation of Host OS and Install QEMU from source

The QEMU emulator is suggested to have this environment up and running using the version of 6.2 and fetch it directly from GitHub to avoid incompatibility issues.

You can fetch and build the QEMU in any OS, but the following instructions are to run it on Ubuntu 20.04 LTS.

```
:~$ cd ~

:~$ sudo apt install ninja-build u-boot-qemu opensbi

:~$ sudo apt build-dep qemu

:~$ git clone https://git.qemu.org/git/qemu.git

:~$ cd qemu

:~/qemu$ git checkout v6.2.0
```

```
:~/qemu$ ./configure --target-list=aarch64-softmmu --enable-virtfs

:~/qemu$ make -j4
```

Ubuntu image from official website:

https://cdimage.ubuntu.com/releases/20.04/release/ubuntu-20.04.3-live-server-arm64.iso

A file named run.sh in the ~/arm64 directory to be created and put the following lines in that file:

```
run.sh

    sudo ~/qemu/build/aarch64-softmmu/qemu-system-aarch64 -M xlnx-
versal-virt        \
      -machine virtualization=true -machine virt,gic-version=3  \
      -cpu cortex-a72 -smp 2 -m 8192                     \
      -drive if=pflash,format=raw,file=efi.img,readonly=on     \
      -drive if=pflash,format=raw,file=varstore.img         \
      -drive if=virtio,format=qcow2,file=disk.img           \
      -device virtio-scsi-pci,id=scsi0                \
      -object rng-random,filename=/dev/urandom,id=rng0      \
      -device virtio-rng-pci,rng=rng0                 \
      -device virtio-net-pci,netdev=net0                  \
      -netdev user,id=net0,hostfwd=tcp::8022-:22          \
      -nographic
      -drive if=none,id=cd,file=ubuntu-20.04.3-live-server-
arm64.iso,format=raw \
      -device scsi-cd,drive=cd
```

Before running the run.sh file, virtual file systems to be created with following commands:

```
:~/arm64$ ~/qemu/build/qemu-img create -f qcow2 disk.img 100G

:~/arm64$ truncate -s 64m varstore.img

:~/arm64$ truncate -s 64m efi.img

:~/arm64$ dd if=/usr/share/qemu-efi-aarch64/QEMU_EFI.fd
of=efi.img conv=notrunc
```

run.sh file to start installing Ubuntu in this environment to be used.

Xilinx PetaLinux

The PetaLinux Tools offers everything necessary to customize, build and deploy Embedded Linux solutions on Xilinx processing systems. Tailored to accelerate design productivity, the solution works with the Xilinx hardware design tools to ease the development of Linux systems for Versal.[14]

Install instructions:

https://www.xilinx.com/support/documentation/sw_manuals/petalinux2013_10/ug976-petalinux-installation.pdf

6.2.1.4 Xilinx Vitis AI

The Vitis™ AI development environment is Xilinx's development platform for AI inference on Xilinx hardware platforms, including both edge devices and Alveo™ cards. It consists of optimized IP, tools, libraries, models, and example designs. It is designed with high efficiency and ease-of-use in mind, unleashing the full potential of AI acceleration on Xilinx FPGA and ACAP. [15]

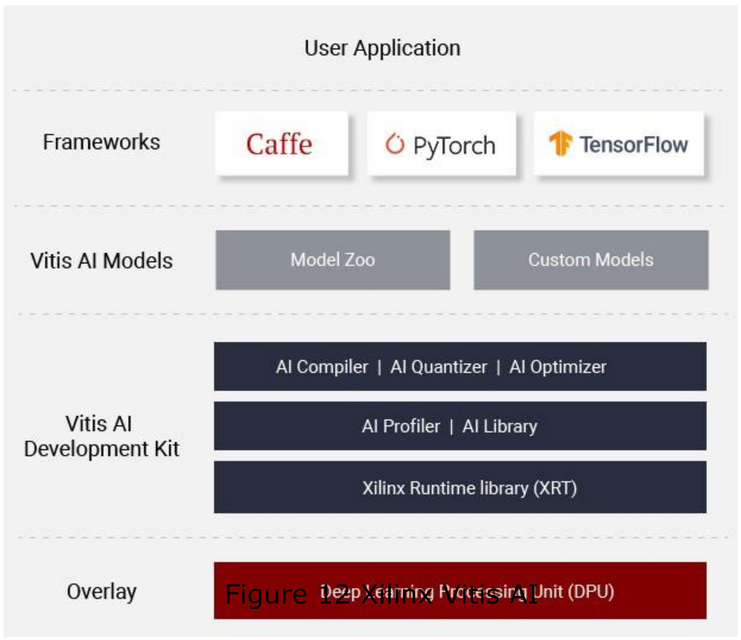


Figure 12: Xilinx Vitis AI

(Image Source: https://www.xilinx.com/products/design-tools/vitis/vitis-ai.html)

Xilinx Vitis AI can alternatively be installed as a microservice from Docker Hub:

```
docker pull paroque28/vitis_ai_build
```

or from local file:

14 https://www.xilinx.com/products/design-tools/embedded-software/petalinux-sdk.html#tools
15 https://www.xilinx.com/products/design-tools/vitis/vitis-ai.html

```
docker load or vitis_ai_build.tar
```

Install Xilinx Vitis AI from Xilinx not as a microservice:

https://www.xilinx.com/htmldocs/xilinx2019_2/vitis_doc/lib_install.html

Versal Emulation: VCK190

VCK-190 is an evaluation kit for developers to design and implement compute-intensive applications due to its AI and DSP engines. The inside CPU of this board has a Dual-Core ARM Cortex-A72 application processing unit which is responsible for running the operating system on this board.

However, the cost of this board is more than 13,000 Euros, and the delivery lead time to get the board is 20 weeks, according to the Xilinx website. So, the easiest way to investigate the functionality of required applications on such a board is to emulate it in a fast manner.

Two ways for the emulation of the Versal board have been explored. As explained in the ARM emulation section, the board has been run QEMU. The other method was to use Nvidia Nano Jetson devices to mimic the application environment of the VCK190 since the architecture of the CPUs is the same.

The following section introduces a straightforward method of running the Vitis AI Core, which is the heart of computation to the VERSAL.

The Dockerized version of the Vitis-AI Core can be used for the sake of simplicity with recommended disk space for Vitis is at least 100GB.

```
:~$ git clone --recurse-submodules https://github.com/Xilinx/Vitis-AI
:~$ docker pull paroque28/vitis_ai_build
:~$ cd Vitis-AI
```

For ARM architecture, you should change a line in docker_run.sh file in Vitis-AI directory.

OLD docker_run.sh	NEW docker_run.sh
… docker_run_params=$(cat <<-END -v /dev/shm:/dev/shm \\ -v /opt/xilinx/dsa:/opt/xilinx/dsa \\	… docker_run_params=$(cat <<-END -v /dev/shm:/dev/shm \\ -v /opt/xilinx/dsa:/opt/xilinx/dsa \\

```
    -v
/opt/xilinx/overlaybins:/opt/xili
nx/overlaybins \

    -e USER=$user -e UID=$uid -e
GID=$gid \

    -e VERSION=$VERSION \

    -v
$DOCKER_RUN_DIR:/vitis_ai_home \

    -v $HERE:/workspace \

    -w /workspace \

    --rm \

    --network=host \

    ${DETACHED} \

    ${RUN_MODE} \

    $IMAGE_NAME \

    $DEFAULT_COMMAND

END

)

…
```

```
    -v
/opt/xilinx/overlaybins:/opt/xili
nx/overlaybins \

    -e USER=$user -e UID=$uid -e
GID=$gid \

    -e VERSION=$VERSION \

    -v
$DOCKER_RUN_DIR:/vitis_ai_home \

    -v $HERE:/workspace \

    -w /workspace \

    --rm \

    --network=host \

    ${DETACHED} \

    ${RUN_MODE} \

    paroque28/vitis_ai_build  \

    $DEFAULT_COMMAND

END

)

…
```

After those changes, the Vitis-AI can be run in the docker_run.sh bash file in the terminal, granting you access to the container terminal.

6.2.2 Orchestration layer

Note, that Mosquitto, Microk8S, Juju are a part of an orchestration layer and not in scope of D6.1. Therefore, only the installation and verifying the installation has been done.

In Kubernetes family there are other installation candidates to edge node, too. The potential usage of alternatives will be clarified in task 6.2 and in its outcome D6.2.

6.2.2.1 Docker

Docker is a platform as a service product that uses OS-level virtualization to deliver software in packages called containers.

The reasoning behind selecting Docker is that it has proven to work seamlessly with Kubernetes family and is widely used globally.

Install Docker repository:

```
sudo apt-get install ca-certificates curl gnupg  lsb-release
```

Add Docker's official GPG key:

```
Sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg |
sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-
keyring.gpg
```

Install Docker:

```
sudo apt-get install docker-ce docker-ce-cli containerd.io
```

6.2.2.2 Mosquitto

Eclipse Mosquitto is an open source (EPL/EDL licensed) message broker that implements the MQTT protocol versions 5.0, 3.1.1 and 3.1. Mosquitto is lightweight and is suitable for use on all devices from low power single board computers to full servers.[16]

The MQTT protocol provides a lightweight method of carrying out messaging using a publish/subscribe model. This makes it suitable for Internet of Things messaging such as with low power sensors or mobile devices such as phones, embedded computers or microcontrollers.

Mosquitto was chosen based on the benchmark Performance Evaluation of MQTT Broker Servers.[17]

Mosquitto has comprehensive API documentation.

https://mosquitto.org/api/files/mosquitto-h.html

16 https://mosquitto.org/
17 Mishra B. (2018) Performance Evaluation of MQTT Broker Servers. In: Gervasi O. et al. (eds) Computational Science and Its Applications – ICCSA 2018. ICCSA 2018. Lecture Notes in Computer Science, vol 10963. Springer, Cham. https://doi.org/10.1007/978-3-319-95171-3_47

Mosquitto can alternatively be installed as a microservice from Docker Hub:

```
docker pull eclipse-mosquitto
```

or from local file:

```
docker load eclipse-mosquitto.tar
```

6.2.2.3 Microk8S

MicroK8s is a low-ops, minimal production Kubernetes and it is an open-source system for automating deployment, scaling, and management of containerized applications. It provides the functionality of core Kubernetes components, in a small footprint, scalable from a single node to a high-availability production cluster. MicroK8s delivers a lightweight, fully-featured, conformant Kubernetes for IoT devices.[18]

Microk8s can be installed within Ubuntu distribution package or alternatively separately.

MicroK8S was chosen based on the comparison of Kubernetes family products[19]

Note that Microk8S is part of an orchestration layer and not in scope of D6.1. Therefore, only the installation and verifying the installation has been done.

In Kubernetes family there are other installation candidates to edge node, too. The potential usage of alternatives will be clarified in task 6.2 and its outcome D6.2.

Install:

```
sudo snap install microk8s --classic
```

6.2.2.4 Ingress

Ingress is an API object that manages external access to the services in a cluster, typically HTTP. Ingress may provide load balancing, SSL termination and name-based virtual hosting. Ingress was chosen to implementation as it is a part of Kubernetes family products and is embedded in Kubernetes products.[20]

18 https://microk8s.io/
19 https://blog.flant.com/small-local-kubernetes-comparison/
20 https://kubernetes.io/docs/concepts/services-networking/ingress/

Ingress is an add-on to Kubernetes family and installed by enabling it from Kubernetes software.

Install:

```
microk8s enable ingress
```

6.2.2.5 Juju

Juju is a model-driven Operator Lifecycle Manager (OLM). Juju greatly improves the experience of running Kubernetes operators, especially in projects that integrate many operators from different publishers.[21]

Juju is a Charmed Operator Framework, composed of a Charmed Operator Lifecycle Manager and the Charmed Operator SDK. Deploy, integrate, and manage Kubernetes, container and VM-native applications seamlessly across hybrid clouds. Juju drives Day 0 through Day 2 operations in your complex environment.

Juju was chosen to the implementation as it works seamlessly with Kubernetes family.[22]

Juju can alternatively be installed as a microservice from Docker Hub:

```
docker pull jujusolutions/jujubox
```

or from local file

```
docker load jujubox.tar
```

Installation of Juju client not as a microservice:

```
sudo snap install juju --classic
```

Create a controller:

```
juju bootstrap microk8s {controller name,e.g. “Oulu”}
```

Verify the bootstrap process:

```
juju clouds
```

21 https://juju.is/
22 https://ubuntu.com/blog/devops-tools-in-2020-why-consider-juju

6.2.2.6 Cilium

Cilium is an open-source software for providing, securing and observing network connectivity between container workloads - cloud native, and fueled by the revolutionary Kernel technology eBPF.

Cilium is open-source software for providing and transparently securing network connectivity and load balancing between application workloads such as application containers or processes. Cilium operates at Layer 3/4 to provide traditional networking and security services as well as Layer 7 to protect and secure use of modern application protocols such as HTTP, gRPC and Kafka.[23]

Cilium was chosen as it has been integrated into Kubernetes orchestration framework.[24]

Cilium can alternatively be installed as a microservice from Docker Hub:

```
docker pull cilium/cilium
```

or from local file:

```
docker load cilium.tar
```

6.2.2.7 Prometheus

Prometheus is a systems and service monitoring system. It collects metrics from configured targets at given intervals, evaluates rule expressions, displays the results, and can trigger alerts if some condition is observed to be true.[25]

Prometheus' main distinguishing features as compared to other monitoring systems are:

- a multi-dimensional data model (timeseries defined by metric name and set of key/value dimensions)
- a flexible query language to leverage this dimensionality
- no dependency on distributed storage; single server nodes are autonomous
- timeseries collection happens via a pull model over HTTP
- pushing timeseries is supported via an intermediary gateway
- targets are discovered via service discovery or static configuration
- multiple modes of graphing and dashboarding support
- support for hierarchical and horizontal federation

23 https://cilium.io/
24 https://kubernetes.io/docs/tasks/administer-cluster/network-policy-provider/cilium-network-policy/
25 https://prometheus.io/

Prometheus was chosen based on the comparison of top 10 Prometheus Alternatives & Competitors.[26]

Prometheus can alternatively be installed as a microservice from Docker Hub:

```
docker pull prom/prometheus
```

or from local file:

```
docker load Prometheus.tar
```

6.2.2.8 Velero

Velero is an open-source tool to safely backup and restore, perform disaster recovery, and migrate Kubernetes cluster resources and persistent volumes.[27]

Velero was chosen based on the comparison Velero alternatives [28]

Velero can alternatively be installed as a microservice from Docker Hub:

```
docker pull velero/velero
```

or from local file:

```
docker load velero.tar
```

6.2.2.9 Istio

Istio extends Kubernetes to establish a programmable, application-aware network using the powerful Envoy service proxy. Working with both Kubernetes and traditional workloads, Istio brings standard, universal traffic management, telemetry, and security to complex deployments has been built on Envoy.[29]

Istio was chosen based on a Comparison of Kubernetes service mesh alternatives[30]

Istio can alternatively be installed as a microservice from Docker Hub:

```
docker pull istio/app_sidecar_ubuntu_focal
```

26 https://www.g2.com/products/prometheus/competitors/alternatives
27 https://velero.io/
28 https://www.baculasystems.com/velero-competitors-alternatives/
29 https://istio.io/
30 https://platform9.com/blog/kubernetes-service-mesh-a-comparison-of-istio-linkerd-and-consul/ and https://techbeacon.com/app-dev-testing/9-open-source-service-meshes-compared

or from local file:

```
docker load app_sidecar_ubuntu_focal.tar
```

6.2.2.10 Envoy

Envoy is an open-source edge and service proxy, designed for cloud-native applications.[31]

Envoy was chosen based on the simplification of the microservice architecture.[32]

Envoy can alternatively be installed as a microservice from Docker Hub:

```
docker pull envoyproxy/envoy
```

or from local file:

```
docker load envoy.tar
```

6.2.2.11 Fluent Bit

Fluent Bit is an open-source Log Processor and Forwarder which allows you to collect any data like metrics and logs from different sources, enrich them with filters and send them to multiple destinations and it comes with full support for Kubernetes.[33]

Fluent Bit was chosen based on log collectors' benchmark.[34]

Fluent Bit can alternatively be installed as a microservice from Docker Hub:

```
docker pull fluent/fluent-bit
```

or from local file:

```
docker load fluent-bit.tar
```

31 https://www.envoyproxy.io/
32 https://samirbehara.com/2018/09/05/simplifying-microservice-architecture-with-envoy-and-istio/
33 https://fluentbit.io/
34 https://medium.com/ibm-cloud/log-collectors-performance-benchmarking-8c5218a08fea

6.2.3 Service/application layer

6.2.3.1 Kubeflow

Kubeflow is the machine learning toolkit for Kubernetes making deployments of machine learning (ML) workflows on Kubernetes simple, portable and scalable.[35]

Kubeflow was chosen by two different factors: it is an integrated part of Kubernetes and was created originally for TensorFlow usage.

To install Kubeflow on Microk8s enabled environment command:

```
microk8S enable Kubeflow
```

6.2.3.2 TensorFlow

TensorFlow is an end-to-end open-source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries, and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML-powered applications.[36]

TensorFlow is ONNX[37] compatible has been identified to meet cognitive and autonomous node requirements for generation of the inference from use cases described in FRACTAL D2.1, p 17 and therefore chosen to be a part of implementation example. Also, Vitis AI framework utilizes TensorFlow

TensorFlow can alternatively be installed as a microservice from Docker Hub::

```
docker pull armswdev/tensorflow-arm-neoverse
```

or from local file:

```
docker load tensorflow-arm-neoverse.tar
```

35 https://www.kubeflow.org/
36 https://www.tensorflow.org/
37 https://onnx.ai/

6.2.3.3 Caffe

Caffe is a deep learning framework made with expression, speed, and modularity in mind. It is developed by Berkeley AI Research (BAIR) and by community contributors.

Caffe has been identified to meet cognitive and autonomous node requirements for generation of the inference from use cases described in FRACTAL D2.1, p 17 and therefore chosen to be a part of implementation example. Also, Vitis AI framework utilizes Caffe

Caffe can alternatively be installed as a microservice from Docker Hub:

```
docker pull bvlc/caffe
```

or from local file:

```
docker load caffe.tar
```

6.2.3.4 Jypyter

The Jupyter Notebook is a web-based interactive computing platform. The notebook combines live code, equations, narrative text, visualizations.[38]

Jupyter was chosen based on the node requirements for notebook tools for data science in FRACTAL D2.1, p 99.

Jupyter can alternatively be installed as a microservice from Docker Hub:

```
docker pull jupyter/datascience-notebook
```

or from local file:

```
docker load datascience-notebook.tar
```

38 https://jupyter.org/

6.2.3.5 PyTorch

PyTorch An open-source machine learning framework and an optimized tensor library for deep learning using GPUs and CPUs that accelerates the path from research prototyping to production deployment[39]

PyTorch was chosen based on the node requirements tools for the generation of ML, DL models in FRACTAL D2.1, p 99. Also, Vitis AI framework utilizes PyTorch

PyTorch can alternatively be installed as a microservice from Docker Hub:

```
docker pull pytorch/pytorch
```

or from local file:

```
docker load pytorch.tar
```

6.2.3.6 OpenCV

OpenCV (Open-Source Computer Vision Library) is an open-source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products[40]

OpenCV was chosen based on Cognitive and autonomous node requirements and the requirement of elaborating data collected using heterogeneous techniques and FRACTAL D2.1, p 43 & 99.

OpenCV can alternatively be installed as a microservice from Docker Hub::

```
docker pull pachyderm/opencv
```

or from local file:

```
docker load opencv
```

39 https://pytorch.org/
40 https://opencv.org/

6.2.3.7 Kafka

Apache Kafka is an open-source distributed event streaming platform used by thousands of companies for high-performance data pipelines, streaming analytics, data integration, and mission-critical applications.

Kafka was chosen based on the node requirements tools for the generation of ML, DL models in FRACTAL D2.1, p 111.

Kafka can alternatively be installed as a microservice from Docker Hub::

```
docker pull confluentinc/cp-kafka
```

or from local file

```
docker load cp-kafka
```

More detailed info about the installation:

https://docs.confluent.io/platform/current/installation/docker/installation.html

6.2.3.8 MongoDB

MongoDB is a source-available cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents with optional schemas.[41]

MongoDB was chosen based on the simplicity and full combability with JSON data model.[42]

An example of the data model can be found from https://json.org/example.html

MongoDB can alternatively be installed as a microservice from Docker Hub:

```
docker pull mongo/mongo
```

or from local file

```
docker load mongo.tar
```

41 https://www.mongodb.com/
42 https://docs.mongodb.com/guides/server/introduction/

6.2.3.9 Spark

Apache Spark is an open-source unified analytics engine for large-scale data processing. Spark provides an interface for programming clusters with implicit data parallelism and fault tolerance.[43]

Spark was chosen based on the node requirements tools for the generation of ML, DL models in FRACTAL D2.1, p 99.

Spark can alternatively be installed as a microservice from Docker Hub::

```
docker pull bitnami/spark
```

or from local file:

```
docker load spark.tar
```

6.2.3.10 ONNX

ONNX Runtime is a cross-platform inference and training machine-learning accelerator for machine learning models with mult- platform support and a flexible interface to integrate with hardware-specific libraries. ONNX Runtime can be used with models from PyTorch, Tensorflow/Keras, TFLite, scikit-learn, and other frameworks.[44]

ONNX Runtime was chosen based on Cognitive and autonomous node requirements and the requirement of elaborating data collected using heterogeneous techniques and FRACTAL D2.1, p 43.

ONNX can alternatively be installed as a microservice from Docker Hub:

```
onnx/onnx-dev
```

or from local file:

```
docker load onnx-dev.tar
```

6.2.3.11 Darknet

Darknet is an open-source neural network framework written in C and CUDA. It is fast, easy to install, and supports CPU and GPU computation.[45]

Darknet was chosen based on Cognitive and autonomous node requirements and the requirement of elaborating data collected using heterogeneous techniques and FRACTAL D2.1, p 38.

43 https://spark.apache.org/
44 https://onnxruntime.ai/
45 Redmon, J., 2013-2016, Darknet: Open Source Neural Networks in C, http://pjreddie.com/darknet

Darknet can alternatively be installed as a microservice from Docker Hub:

```
docker pull daisukekobayashi/darknet
```

or from local file:

```
docker load darknet
```

6.2.3.12 Keras

Keras is the most used deep learning framework. Keras makes it easier to run new experiments.[46]

Keras was chosen based on Cognitive and autonomous node requirements and the requirement of elaborating data collected using heterogeneous techniques and FRACTAL D2.1, p 49.

Keras can alternatively be installed as a microservice from Docker Hub:

```
docker pull ermaker/keras
```

or from local file:

```
docker load keras.tar
```

6.2.3.13 Chainer

Chainer is a Python-based deep learning framework aiming at flexibility. It provides automatic differentiation APIs based on the define-by-run approach (a.k.a. dynamic computational graphs) as well as object-oriented high-level APIs to build and train neural networks

Chainer was chosen based on Cognitive and autonomous node requirements and the requirement of elaborating data collected using heterogeneous techniques.

Chainer can alternatively be installed as a microservice from Docker Hub:

```
docker pull chainer/chainer
```

or from local file

46 https://keras.io/

```
docker load chainer.tar
```

6.2.3.14 Undefined microservices

One of the design principles of the architecture has been that new microservices can be managed through the whole life-cycle deployed, commissioned, and decommissioned easy and manageable way. This gives room in the architecture to containerize and implement new services needed to fulfill the future use need. Docker documentation describes how microservices are built for Docker.

6.3 Mid-range node (RISC-V64)

6.3.1 Middleware layer

The middleware layer is mostly a part of WP3 and described in D3.2 Preliminary Fractal Software node and services, but to enable the implementation of the architecture the middleware installation is required.

Also, Ubuntu offers ready packaged Software/orchestration layer tools such as Microk8s, Docker, Mosquitto, Prometheus and Juju to be installed simultaneously with the operating system installation.

6.3.1.1 Operating system

The needed normal pre-installed Ubuntu server for RISC-V.

Ubuntu:
https://cdimage.ubuntu.com/ubuntu/releases/21.04/release/ubuntu-21.04-preinstalled-server-riscv64+unleashed.img.xz

Debian:

https://gitlab.com/api/v4/projects/giomasce%2Fdqib/jobs/artifacts/master/download?job=convert_riscv64-virt

6.3.1.2 RISC-V64 Emulation

The Ubuntu/Debian Operating System can be installed in RISC-V64 architecture on real bare metal or emulated environments. Demonstration of the emulation version has been done for the sake of the simplicity and availability of the system. Note that the emulated hardware is slower than the actual one, so the suggestion is to use more RAM and find the fine-tuned number of cores for the CPU (see the example below).

For RISC-V the QEMU emulator is suggested to have this environment up and running using the version of 6.2 and to fetch it directly from GitHub to avoid incompatibility issues.

Preparing the Host OS and Installing QEMU

You can fetch and build the QEMU in various operating systems, but the following instructions are to run it on Ubuntu 20.04 LTS.

```
:~$ cd ~

:~$ sudo apt install ninja-build u-boot-qemu opensbi

:~$ sudo apt build-dep qemu

:~$ git clone https://git.qemu.org/git/qemu.git

:~$ cd qemu

:~/qemu$ git checkout v6.2.0

:~/qemu$ ./configure --target-list=riscv64-softmmu --enable-virtfs

:~/qemu$ make -j4
```

Any pre-built image can be downloaded from Ubuntu official website from the following link. However, for this example, we used Debian image, which, from the following commands, you can have the same.

Source to other images: https://wiki.ubuntu.com/RISC-V

```
:~$ cd ~

:~$ sudo apt install wget

:~$ wget \

"https://gitlab.com/api/v4/projects/giomasce%2Fdqib/jobs/artifacts/mas
ter/download?job=convert_riscv64-virt" -O debian-rv64.zip

:~$ mkdir debian-rv64

:~$ cd debian-rv64

:~/debian-rv64$ unzip ../debian-rv64.zip

:~/debian-rv64$ cd artifacts

# (OPTIONAL) You can resize the image file
```

```
:~/debian-rv64/artifacts$ qemu-img resize -f qcow2 image.qcow2 +15G
```

Now, create a file named run.sh in the ~/debian-rv64/artifacts directory and put the following lines in that file:

run.sh

```
sudo ~/qemu/build/riscv64-softmmu/qemu-system-riscv64 \
  -machine virt \
  -cpu rv64 \
  -m 4G \
  -smp 4 \
  -device virtio-blk-device,drive=hd \
  -drive file=overlay.qcow2,if=none,id=hd \
  -device virtio-net-device,netdev=net \
  -netdev user,id=net,hostfwd=tcp::2222-:22 \
  -bios /usr/lib/riscv64-linux-gnu/opensbi/generic/fw_jump.elf \
  -kernel /usr/lib/u-boot/qemu-riscv64_smode/uboot.elf \
  -object rng-random,filename=/dev/urandom,id=rng \
  -device virtio-rng-device,rng=rng \
  -append "root=LABEL=rootfs console=ttyS0" \
  -nographic
```

Then, follow the next commands to run the emulation and make the system UP and Running.

```
:~/debian-rv64/artifacts$ chmod +x run.sh
:~/debian-rv64/artifacts$ ./run.sh
```

System can be logged in with root username, and the default password is also root.

6.3.2 Orchestration layer

Note, that Mosquitto, K3S and Juju are a part of an orchestration layer and not in scope of D6.1. Therefore, only the installation and verifying the installation has been done.

In Kubernetes family there are other installation candidates to edge node, too. The potential usage of alternatives will be clarified in task 6.2 and in its outcome D6.2.

6.3.2.1 Docker

Docker is a platform as a service product that uses OS-level virtualization to deliver software in packages called containers.

The reasoning behind selecting Docker is that it has proven to work seamlessly with Kubernetes family and is widely used globally.

The RISC-V environment does not have a native docker repository to be installed, so you should install the requirements for Docker first, then build the Docker manually for the OS.[47]

Run the following commands in your QEMU RISC-V environment to install Go:

```
# Download the tarball into the VM
wget https://github.com/carlosedp/riscv-
bringup/releases/download/v1.0/go-1.16.7-riscv64.tar.gz
# In the VM, unpack (in root dir for example)
tar vxf go-1.16.7-riscv64.tar.gz -C /usr/local
# Add to your PATH
export PATH="/usr/local/go/bin:$PATH"
# Add to bashrc
echo "export PATH=/usr/local/go/bin:$PATH" >> ~/.bashrc
```

Then Docker is installed using the following commands and after reboot, check and validate the installation of Docker with hello-world!

```
wget https://github.com/carlosedp/riscv-
bringup/releases/download/v1.0/docker-20.10.2-dev_riscv64.deb
sudo apt install ./docker-20.10.2-dev_riscv64.deb
```

After rebooting to validate the installation:

```
$ docker run hello-world
```

6.3.2.2 Installation Candidates

RISC-V environment is under development by adding native libraries for this system to have the tools and application by running `apt install` commands. This means that most of the tools for this system should be fetched and built manually. For example, it is possible to install PostgreSQL database in this system running `apt install postgresql` command.

Recommendation is to install the tools and application in the RISC-V environment with Docker. To illustrate a simple example in Task 6.1 the lightweight Docker image of PostgreSQL was built and pushed it into the Docker hub, so there is no need to extra work on building the tools (which is available at this address https://hub.docker.com/r/vahidm/postgres). Although this image can be pulled and run it using an internet connection, in the scenarios, without an internet connection,

47 https://carlosedp.medium.com/docker-containers-on-risc-v-architecture-5bc45725624b

you can transfer these image files in flash memory, hard disks, local network access, etc., and run them on those OSs manually.

A simple general example of this procedure is as follows:

1. Save the Docker running container (which you are already sure that is working fine) with the following command:

   ```
   $ docker save my-img -> my-img-to-transfer.tar
   ```

2. Copy .tar file in any shareable material.
3. Transfer!
4. Load and run the image file in other OS:

   ```
   $ docker load < my-img-to-transfer.tar
   ```

   ```
   $ docker run my-img
   ```

Kubernetes or K3s almost are possible to build the same way building the Docker. RISC-V nodes can be connected to the Kubernetes cluster for management and orchestration purposes.

6.3.2.3 Mosquitto

Eclipse Mosquitto is an open source (EPL/EDL licensed) message broker that implements the MQTT protocol versions 5.0, 3.1.1 and 3.1. Mosquitto is lightweight and is suitable for use on all devices from low power single board computers to full servers.[48]

The MQTT protocol provides a lightweight method of carrying out messaging using a publish/subscribe model. This makes it suitable for Internet of Things messaging such as with low power sensors or mobile devices such as phones, embedded computers or microcontrollers.

Mosquitto was chosen based on the benchmark Performance Evaluation of MQTT Broker Servers.[49]

Mosquitto has comprehensive API documentation.

https://mosquitto.org/api/files/mosquitto-h.html

Mosquitto can alternatively be installed as a microservice from Docker Hub:

48 https://mosquitto.org/

49 Mishra B. (2018) Performance Evaluation of MQTT Broker Servers. In: Gervasi O. et al. (eds) Computational Science and Its Applications – ICCSA 2018. ICCSA 2018. Lecture Notes in Computer Science, vol 10963. Springer, Cham. https://doi.org/10.1007/978-3-319-95171-3_47

```
docker pull eclipse-mosquitto
```

or from local file:

```
docker load eclipse-mosquitto.tar
```

6.3.2.4 K3S

K3S is a highly available, certified Kubernetes distribution designed for production workloads in unattended, resource-constrained, remote locations or inside IoT

K3S was chosen based on the comparison of Kubernetes family products[50]

Note that K3S is a part of an orchestration layer and not in scope of D6.1. Therefore, only the installation and verifying the installation has been done.

In Kubernetes family there are other installation candidates to edge node, too. The potential usage of alternatives will be clarified in task 6.2 and its outcome D6.2.

Install (in the RISC-V machine):

```
~/$ wget https://github.com/carlosedp/riscv-
bringup/releases/download/v1.0/k3s-v1.20.4-k3s1-riscv64.tar.gz

~/$ mkdir install-k3s

~/$ tar xvf k3s-v1.20.4-k3s1-riscv64.tar.gz -C install-k3s

~/$ cd install-k3s

~/install-k3s$ ./install.sh
```

To validate the installation:

```
~/$ k3s kubectl get node
```

6.3.2.5 Ingress

Ingress is an API object that manages external access to the services in a cluster, typically HTTP. Ingress may provide load balancing, SSL termination and name-based virtual hosting. Ingress was chosen to implementation as it is a part of Kubernetes family products and is embedded in Kubernetes products.[51]

50 https://blog.flant.com/small-local-kubernetes-comparison/
51 https://kubernetes.io/docs/concepts/services-networking/ingress/

Ingress is an add-on to Kubernetes family and installed by enabling it from Kubernetes software.

Install:

```
kubectl apply -f
https://raw.githubusercontent.com/kubernetes/ingress-
nginx/controller-
v0.47.0/deploy/static/provider/baremetal/deploy.yaml
```

6.3.2.6 Juju

Juju is a model-driven Operator Lifecycle Manager (OLM). Juju greatly improves the experience of running Kubernetes operators, especially in projects that integrate many operators from different publishers.[52]

Juju is a Charmed Operator Framework, composed of a Charmed Operator Lifecycle Manager and the Charmed Operator SDK. Deploy, integrate, and manage Kubernetes, container and VM-native applications seamlessly across hybrid clouds. Juju drives Day 0 through Day 2 operations in your complex environment.

Juju was chosen to the implementation as it works seamlessly with Kubernetes family.[53]

Juju can alternatively be installed as a microservice from Docker Hub:

```
docker pull jujusolutions/jujubox
```

or from local file

```
docker load jujubox.tar
```

Installation of Juju client not as a microservice:

```
sudo snap install juju --classic
```

Create a controller:

```
juju bootstrap K3S {controller name,e.g. “Oulu”}
```

52 https://juju.is/
53 https://ubuntu.com/blog/devops-tools-in-2020-why-consider-juju

Verify the bootstrap process:

```
juju clouds
```

6.3.2.7 Cilium

Cilium is an open-source software for providing, securing and observing network connectivity between container workloads - cloud native, and fueled by the revolutionary Kernel technology eBPF.

Cilium is open-source software for providing and transparently securing network connectivity and load balancing between application workloads such as application containers or processes. Cilium operates at Layer 3/4 to provide traditional networking and security services as well as Layer 7 to protect and secure use of modern application protocols such as HTTP, gRPC and Kafka.[54]

Cilium was chosen as it has been integrated into Kubernetes orchestration framework.[55]

Cilium can alternatively be installed as a microservice from Docker Hub:

```
docker pull cilium/cilium
```

or from local file:

```
docker load cilium.tar
```

6.3.2.8 Prometheus

Prometheus is a systems and service monitoring system. It collects metrics from configured targets at given intervals, evaluates rule expressions, displays the results, and can trigger alerts if some condition is observed to be true.[56]

Prometheus' main distinguishing features as compared to other monitoring systems are:

- a multi-dimensional data model (timeseries defined by metric name and set of key/value dimensions)
- a flexible query language to leverage this dimensionality
- no dependency on distributed storage; single server nodes are autonomous

54 https://cilium.io/
55 https://kubernetes.io/docs/tasks/administer-cluster/network-policy-provider/cilium-network-policy/
56 https://prometheus.io/

- timeseries collection happens via a pull model over HTTP
- pushing timeseries is supported via an intermediary gateway
- targets are discovered via service discovery or static configuration
- multiple modes of graphing and dashboarding support
- support for hierarchical and horizontal federation

Prometheus was chosen based on the comparison of top 10 Prometheus Alternatives & Competitors.[57]

Prometheus can alternatively be installed as a microservice from Docker Hub:

```
docker pull prom/prometheus
```

or from local file:

```
docker load Prometheus.tar
```

6.3.2.9 Velero

Velero is an open-source tool to safely backup and restore, perform disaster recovery, and migrate Kubernetes cluster resources and persistent volumes.[58]

Velero was chosen based on the comparison Velero alternatives [59]

Velero can alternatively be installed as a microservice from Docker Hub:

```
docker pull velero/velero
```

or from local file:

```
docker load velero.tar
```

6.3.2.10 Istio

Istio extends Kubernetes to establish a programmable, application-aware network using the powerful Envoy service proxy. Working with both Kubernetes and traditional workloads, Istio brings standard, universal traffic management, telemetry, and security to complex deployments has been built on Envoy.[60]

57 https://www.g2.com/products/prometheus/competitors/alternatives
58 https://velero.io/
59 https://www.baculasystems.com/velero-competitors-alternatives/
60 https://istio.io/

Istio was chosen based on a Comparison of Kubernetes service mesh alternatives[61]

Istio can alternatively be installed as a microservice from Docker Hub:

```
docker pull istio/app_sidecar_ubuntu_focal
```

or from local file:

```
docker load app_sidecar_ubuntu_focal.tar
```

6.3.2.11 Envoy

Envoy is an open-source edge and service proxy, designed for cloud-native applications.[62]

Envoy was chosen based on the simplification of the microservice architecture.[63]

Envoy can alternatively be installed as a microservice from Docker Hub:

```
docker pull envoyproxy/envoy
```

or from local file:

```
docker load envoy.tar
```

6.3.2.12 Fluent Bit

Fluent Bit is an open-source Log Processor and Forwarder which allows you to collect any data like metrics and logs from different sources, enrich them with filters and send them to multiple destinations and it comes with full support for Kubernetes.[64]

Fluent Bit was chosen based on log collectors' benchmark.[65]

Fluent Bit can alternatively be installed as a microservice from Docker Hub:

```
docker pull fluent/fluent-bit
```

or from local file:

```
docker load fluent-bit.tar
```

61 https://platform9.com/blog/kubernetes-service-mesh-a-comparison-of-istio-linkerd-and-consul/ and https://techbeacon.com/app-dev-testing/9-open-source-service-meshes-compared
62 https://www.envoyproxy.io/
63 https://samirbehara.com/2018/09/05/simplifying-microservice-architecture-with-envoy-and-istio/
64 https://fluentbit.io/
65 https://medium.com/ibm-cloud/log-collectors-performance-benchmarking-8c5218a08fea

6.3.3 Service/application layer

6.3.3.1 Kubeflow

Kubeflow is the machine learning toolkit for Kubernetes making deployments of machine learning (ML) workflows on Kubernetes simple, portable and scalable.[66]

Kubeflow was chosen by two different factors: it is an integrated part of Kubernetes and was created originally for TensorFlow usage.

To install Kubeflow on K3S enabled environment command:

```
git clone https://github.com/kubeflow/manifests.git

$ cd example
$ ls
kustomization.yaml

kustomize build | kubectl apply -f -
kubectl edit -n kubeflow gateways.networking.istio.io kubeflow-
gateway
```

6.3.3.2 TensorFlow

TensorFlow is an end-to-end open-source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries, and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML-powered applications.[67]

TensorFlow is ONNX[68] compatible has been identified to meet cognitive and autonomous node requirements for generation of the inference from use cases described in FRACTAL D2.1, p 17 and therefore chosen to be a part of implementation example. Also, Vitis AI framework utilizes TensorFlow

66 https://www.kubeflow.org/
67 https://www.tensorflow.org/
68 https://onnx.ai/

TensorFlow can alternatively be installed as a microservice from Docker Hub::

```
docker pull armswdev/tensorflow-arm-neoverse
```

or from local file:

```
docker load tensorflow-arm-neoverse.tar
```

6.3.3.3 Caffe

Caffe is a deep learning framework made with expression, speed, and modularity in mind. It is developed by Berkeley AI Research (BAIR) and by community contributors.

Caffe has been identified to meet cognitive and autonomous node requirements for generation of the inference from use cases described in FRACTAL D2.1, p 17 and therefore chosen to be a part of implementation example. Also, Vitis AI framework utilizes Caffe

Caffe can alternatively be installed as a microservice from Docker Hub:

```
docker pull bvlc/caffe
```

or from local file:

```
docker load caffe.tar
```

6.3.3.4 Jypyter

The Jupyter Notebook is a web-based interactive computing platform. The notebook combines live code, equations, narrative text, visualizations.[69]

Jupyter was chosen based on the node requirements for notebook tools for data science in FRACTAL D2.1, p 99.

Jupyter can alternatively be installed as a microservice from Docker Hub:

```
docker pull jupyter/datascience-notebook
```

or from local file:

```
docker load datascience-notebook.tar
```

69 https://jupyter.org/

6.3.3.5 PyTorch

PyTorch An open-source machine learning framework and an optimized tensor library for deep learning using GPUs and CPUs that accelerates the path from research prototyping to production deployment[70]

PyTorch was chosen based on the node requirements tools for the generation of ML, DL models in FRACTAL D2.1, p 99. Also, Vitis AI framework utilizes PyTorch

PyTorch can alternatively be installed as a microservice from Docker Hub:

```
docker pull pytorch/pytorch
```

or from local file:

```
docker load pytorch.tar
```

6.3.3.6 OpenCV

OpenCV (Open-Source Computer Vision Library) is an open-source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products[71]

OpenCV was chosen based on Cognitive and autonomous node requirements and the requirement of elaborating data collected using heterogeneous techniques and FRACTAL D2.1, p 43 & 99.

OpenCV can alternatively be installed as a microservice from Docker Hub::

```
docker pull pachyderm/opencv
```

or from local file:

```
docker load opencv
```

70 https://pytorch.org/
71 https://opencv.org/

6.3.3.7 Kafka

Apache Kafka is an open-source distributed event streaming platform used by thousands of companies for high-performance data pipelines, streaming analytics, data integration, and mission-critical applications.

Kafka was chosen based on the node requirements tools for the generation of ML, DL models in FRACTAL D2.1, p 111.

Kafka can alternatively be installed as a microservice from Docker Hub::

```
docker pull confluentinc/cp-kafka
```

or from local file

```
docker load cp-kafka
```

More detailed info about the installation:

https://docs.confluent.io/platform/current/installation/docker/installation.html

6.3.3.8 postgreSQL

PostgreSQL is a powerful, open-source object-relational database system with over 30 years of active development that has earned it a strong reputation for reliability, feature robustness, and performance.

PostgreSQL can alternatively be installed as a microservice from Docker Hub:

```
docker pull vahidm/postgres
```

or from local file

```
docker load postgres.tar
```

6.3.3.9 Spark

Apache Spark is an open-source unified analytics engine for large-scale data processing. Spark provides an interface for programming clusters with implicit data parallelism and fault tolerance.[72]

72 https://spark.apache.org/

Spark was chosen based on the node requirements tools for the generation of ML, DL models in FRACTAL D2.1, p 99.

Spark can alternatively be installed as a microservice from Docker Hub:

```
docker pull bitnami/spark
```

or from local file:

```
docker load spark.tar
```

6.3.3.10 ONNX

ONNX Runtime is a cross-platform inference and training machine-learning accelerator for machine learning models with multi-platform support and a flexible interface to integrate with hardware-specific libraries. ONNX Runtime can be used with models from PyTorch, Tensorflow/Keras, TFLite, scikit-learn, and other frameworks.[73]

ONNX Runtime was chosen based on Cognitive and autonomous node requirements and the requirement of elaborating data collected using heterogeneous techniques and FRACTAL D2.1, p 43.

ONNX can alternatively be installed as a microservice from Docker Hub:

```
onnx/onnx-dev
```

or from local file:

```
docker load onnx-dev.tar
```

6.3.3.11 Darknet

Darknet is an open-source neural network framework written in C and CUDA. It is fast, easy to install, and supports CPU and GPU computation.[74]

Darknet was chosen based on Cognitive and autonomous node requirements and the requirement of elaborating data collected using heterogeneous techniques and FRACTAL D2.1, p 38.

Darknet can alternatively be installed as a microservice from Docker Hub:

73 https://onnxruntime.ai/
74 Redmon, J., 2013-2016, Darknet: Open Source Neural Networks in C, http://pjreddie.com/darknet

```
docker pull daisukekobayashi/darknet
```

or from local file:

```
docker load darknet
```

6.3.3.12 Keras

Keras is the most used deep learning framework. Keras makes it easier to run new experiments.[75]

Keras was chosen based on Cognitive and autonomous node requirements and the requirement of elaborating data collected using heterogeneous techniques and FRACTAL D2.1, p 49.

Keras can alternatively be installed as a microservice from Docker Hub:

```
docker pull ermaker/keras
```

or from local file:

```
docker load keras.tar
```

6.3.3.13 Chainer

Chainer is a Python-based deep learning framework aiming at flexibility. It provides automatic differentiation APIs based on the define-by-run approach (a.k.a. dynamic computational graphs) as well as object-oriented high-level APIs to build and train neural networks

Chainer was chosen based on Cognitive and autonomous node requirements and the requirement of elaborating data collected using heterogeneous techniques.

Chainer can alternatively be installed as a microservice from Docker Hub:

```
docker pull chainer/chainer
```

or from local file

```
docker load chainer.tar
```

75 https://keras.io/

6.3.3.14 Undefined microservices

One of the design principles of the architecture has been that new microservices can be managed through the whole life-cycle deployed, commissioned, and decommissioned easy and manageable way. This gives room in the architecture to containerize and implement new services needed to fulfill the future use need. Docker documentation describes how microservices are built for Docker.

6.4 Low-end node

6.4.1 Middleware layer

6.4.1.1 Operating system

Apache NuttX is a real time embedded operating system (RTOS) for low-end embedded environment needs.[76]

NuttX installation:

https://nuttx.apache.org/docs/latest/quickstart/install.html

6.4.1.2 NuttX Simulation

Simulated environment for implementation simulation has NuttX own simulator that can run on Linux.

NuttX simulator install:

https://nuttx.apache.org/docs/latest/guides/simulator.html

6.4.2 Application layer

6.4.2.1 C/C++ Libraries

Standard C library fully integrated to NuttX operating system

6.4.2.2 cJSON

cJSON is ultralightweight JSON parser in ANSI C and fully integrated to NuttX operating system

76 https://nuttx.apache.org/docs/latest/introduction/about.html

6.4.2.3 Undefined services

Additional software and services are installed by adding them into NuttX build, compiling it and the flashing it to the device.

6.5 Generic Data model

6.5.1 JSON

JSON (JavaScript Object Notation) is a lightweight, text-based, language-independent syntax for defining data interchange formats. It was derived from the ECMAScript programming language but is programming language independent. JSON defines a small set of structuring rules for the portable representation of structured data.[77] An example of the data model can be found from https://json.org/example.html.

JSON was selected as generics data model implementation as it meets the requirements set in the architecture.

77 https://www.ecma-international.org/publications-and-standards/standards/ecma-404/

7 List of Figures

8 List of Tables