Grant Agreement No.: 101070473
Call: HORIZON-CL4-2021-DATA-01
Topic: HORIZON-CL4-2021-DATA-01-05
Type of action: HORIZON-RIA

D1.3 FLUIDOMOS

Revision: v.1.2

Sub-project Name	FLUIDOMOS
Authors	FLUIDOMOS Team
Submission date	31/01/2025
Reviewers	FLUIDOS Internal

Document Revision History

Version	Date	Description of change	List of contributor(s)
V1.0	20/12/2024	Draft Version	Vemar&ICT
V1.1	02/01/2025	Draft Version - Integrated	Vemar&ICT
V1.2	22/01/2025	Draft Version to be cheked	Vemar&ICT
V2.0	31/01/2025	First Official Emission	Vemar&ICT

Disclaimer

The information, documentation and figures available in this deliverable are written by the Beneficiaries of the "Flexible, scaLable and secUre decentralizeD Operating System" (FLUIDOS) project's Financial Support to Third Parties scheme under EC grant agreement 101070473 and do not necessarily reflect the views of the European Commission.

The European Commission is not liable for any use that may be made of the information contained herein.

Copyright notice

© 2022 - 2025 FLUIDOS Consortium

Funded by Horizon Europe Framework Programme of the European Union

Table Of Contents

Table Of Contents 3
List Of Figures 5
List Of Tables 6
Abbreviations 7
1. Starting Point of the FLUIDOMOS Project 8
2. FLUIDOMOS Overview 10
3. Project Activities And Results 12
Milestone M1 16
KPI 1.1 16
KPI 1.2 17
KPI 1.3 24
KPI 1.4 30
Milestone M2 32
KPI 2.1 32
KPI 2.2 33
KPI 2.3 39
Milestone M3 50
KPI 3.1 50
KPI 3.2 54
KPI 3.3 67
KPI 3.4 67
KPI 3.5 67
FLUIDOMOS Cluster Management 67
Monitoring System 73
Metrics Monitoring Tool 73
Visualization Tool 74
4. Benefits 80
The innovation potential 80
The circular nature of the project 80
Energy Efficiency 80
Economic Savings 81
Quality and Feasibility of the Technical Approach 81
Soundness of the Approach and Credibility 81
Resilience 82
Implementation 83
Benefits and Advantages 83

© 2022-2025 FLUIDOS

Funded by Horizon Europe Framework Programme of the European Union

Optimized Edge Node Utilization..... 84
Conclusion..... 84

Funded by Horizon Europe Framework Programme of the European Union

List Of Figures

Figure 1 : The pilot case study: building automation 9
Figure 2: FLUIDOMOS Multi Cluster Environment Powered By Fluidos 11
Figure 3: FLUIDOMOS project views by using Odoo 16
Figure 4: FLUIDOMOS project tasks 17
Figure 5: The current DELIS architecture 18
Figure 6 The FLUIDOMOS architecture 20
Figure 7 - Clusters topology of FLUIDOMOS Use Case 24
Figure 8: Delis Architecture 24
Figure 9 - DELIS CPU Load 34
Figure 10 - DELIS CPU Utilization 35
Figure 11 - DELIS Used RAM 36
Figure 12 - DELIS Caches 36
Figure 13 -DELIS Active vs Inactive Memory 37
Figure 14 - DELIS RAM Used 38
Figure 15 - DELIS UpTime 38
Figures from 16 to 29 are related to KPI 2.3
Figure 30 - IOT Expansion Modules and Core 51
Figures 31 to 37 are related to KPI 3.2
Figures 38 to 46 are related to Cluster Management
Figures 47 to 56 are related to Monitoring System
Figure 57: The Impact of FLUIDOS on FLUIDOMOS 83

© 2022-2025 FLUIDOS

Funded by Horizon Europe Framework Programme of the European Union

List Of Tables

Table 1: Outcomes 13
Table 2: Project Milestones 14
Table 3: FLUIDOMOS Installation steps 23
Table 4: Delis Services And Plugin 26
Table 5: Delis WS API 28
Table 6: Delis Plugin API 30
Table 7: Accounting records at D1.2 31
Table 8 : FLUIDOMOS Services 32
Table 9 : IOT Components 50
Table 10: Accounting records for all the project activities 68

Funded by Horizon Europe Framework Programme of the European Union

Abbreviations

DED	Dual Entity Device	Complex structure that represents the physical and IT duality of IOT devices
DER	Dual Entity Resource	Complex superstructure representing the physical and IT duality of IOT resources, with ded functions and/or ded group functions
KPI	Key Performances Indicator	
WC	Web and Core	Delis Main Module
WS	Web Services	

© 2022-2025 FLUIDOS

Funded by Horizon Europe Framework Programme of the European Union

1. Starting Point of the FLUIDOMOS Project

The Use Case FLUIDOMOS was conceived to extend the DELIS V2.0 platform, a seamless, reconfigurable, and expandable home/building automation platform, to the FLUIDOS Computing Continuum environment.

The current version of DELIS is a Cloud platform that provides domotic system users (consumers, business, system integrators, and domotic system maintainers) an effective solution for:

- design, implementation, and development
- control and management
- reconfiguration and troubleshooting

of home/building automation systems made by multiple and heterogeneous devices with the least cost, the best scalability, and compatibility with as many possible third-party systems and devices.

The DELIS project has been developed considering that in most cases, even when state-of-the-art technology is used, there are no systems capable of unifying, under a single platform, the entire development process of a home/building automation system that requires a heterogeneous ecosystem of components. The DELIS system is a digital framework that aims to support the roles of designer, installer, home/building automation service provider company, and the end user in each phase of the domotic system life cycle: design, implementation, installation, maintenance of both IoT devices and software for interaction with IoT devices.

DELIS, in summary, is a Cloud platform (https://delis.vemarsas.it/) that exposes a set of microservices, through which one can interface to orchestrate their functionality and build home automation applications using heterogeneous IoT technologies. By abstracting and encapsulating the technical complexities required for IoT interfacing and communication, DELIS's microservices enable interfacing with three different types of IoT data providers:

1 Edge devices, which can be built ad hoc, using common communication protocols and interfaces and general-purpose controllers such as, for example, Raspberry and/or RaZberry. Such Edge devices are intended to monitor and control multiple IOT devices, necessary for home/building automation systems, provided by different Vendors.

2 Edge devices, distributed by different Vendors and ready to use, to take advantage of the same functions exposed in the previous point through standardized interfaces and communication protocols.

3 Cloud platforms, offered by the various home automation technology Vendors, allow monitoring and control of their Edge/IOT technologies through interfacing APIs.

© 2022-2025 FLUIDOS

Funded by Horizon Europe Framework Programme of the European Union

A typical scenario in which the DELIS framework operates is illustrated in Fig. 1, which shows a building automation case with the implementation of a home automation system in a private apartment and, within the same building, an office automation system. As can be seen, the currently implemented architecture reflects the silos-based computing continuum model, already discussed in the "open call Fluidos overview V1.0" white paper. The many drawbacks of the silos model implemented in DELIS led us to submit this proposal.

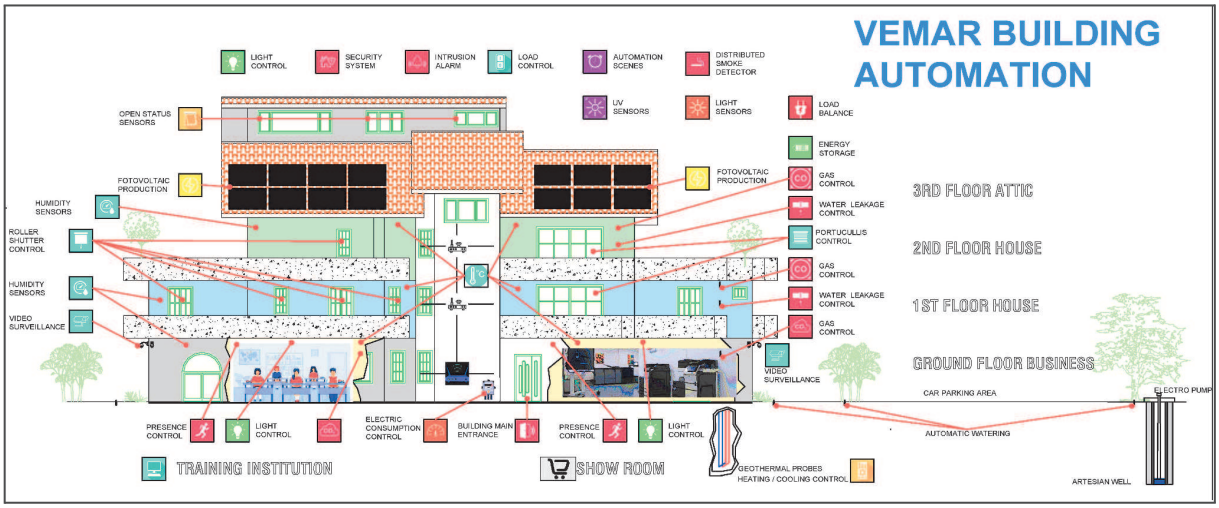


Figure 1 : The pilot case study: building automation

Funded by Horizon Europe Framework Programme of the European Union

2. FLUIDOMOS Overview

The FLUIDOMOS project aims to create a use case that extends the DELIS V2.0 platform (conceived as a seamless, reconfigurable, and expandable home/building automation platform) to the FLUIDOS Computing Continuum environment.
The proposed use case aims to translate the DELIS Cloud architectural model into the continuum computing architectural model of FLUIDOS by exposing DELIS software micro services through the FLUIDOS interface.

There are two benefits achievable for the FLUIDOS ecosystem resulting from the implementation of the FLUIDOMOS use case:

1. The validation of the fluid computing continuum functionality in the home/building automation application scenario.

2. The spread of the FLUIDOS platform to support home/building automation systems, where continuous improvement and state-of-the-art product and service innovations are required.

Regarding the first point, the result deriving from the migration of DELIS (implemented on a cloud architecture) to an architecture that uses the fluid computing continuum paradigm will emerge (FLUIDOMOS). The realization, testing, usage, and performance evaluation of FLUIDOMOS will constitute a validation test of the FLUIDOS platform and its features/functions/services from a use case that can contribute to demonstrating the benefits of using FLUIDOS.

The main purpose of the FLUIDOMOS project is to deploy DELIS microservices via the FLUIDOS meta operating system enabling a scalable FLUIDOS-based orchestrator to control and update a large swarm of IoT devices.

The expected benefits involve the DELIS system, which will be taken toward a higher technological standard, and the FLUIDOS system, which will be validated on a use case with promising application opportunities in the home/building automation scenario.

Technically speaking the FLUIDOMOS project aims to port the current DELIS system from its native Cloud architecture into a Liquid Edge-Cloud architecture. This result can be achieved by updating all types of DELIS microservices in the form of K8S Workers manageable in FLUIDOS, and so with the capability to be orchestrated and distributed, indifferently, in the Cloud or the Meta Edge Nodes.

The following figure shows how the FLUIDOS model will be used to define a particular instance in which DELIS microservices will be deployed in FLUIDOS. The main result of using FLUIDOMOS, compared to the DELIS native design, is that, as long as there is the availability and accessibility to a physical edge computing infrastructure located near the IOT sensors, the availability of home/building

© 2022-2025 FLUIDOS

Funded by Horizon Europe Framework Programme of the European Union

automation services can be guaranteed even when a failure of communication with the Cloud occurs.

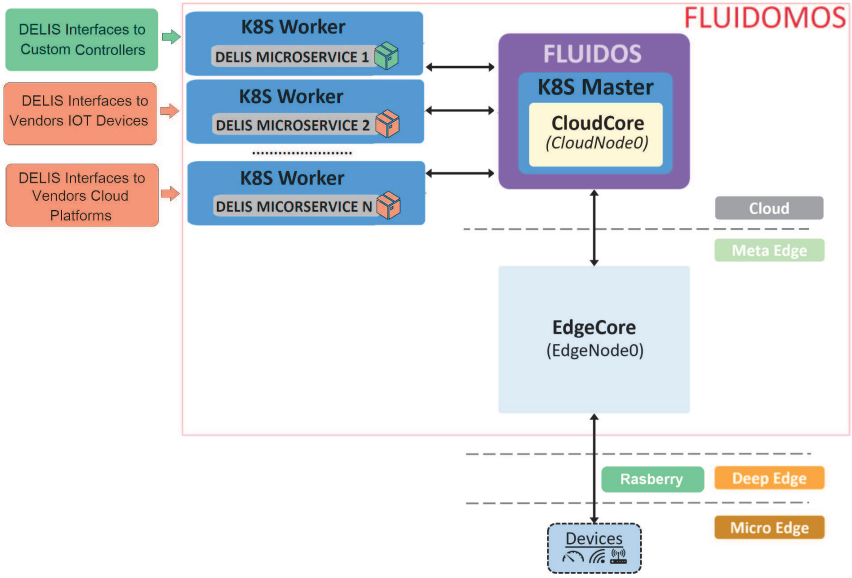


Figure 2: The Deployments Of Fluidomos Services Into A Multi Cluster Environment Powered By Fluidos

Funded by Horizon Europe Framework Programme of the European Union

3. Project Activities And Results

The results we obtained during the development of FLUIDOMOS project can be shortly listed as follows:

1. Project organization by using the SCRUM methodology
2. Understanding of the general concepts of LIQO and FLUIDOS
3. Modelling the new DELIS architecture for FLUIDOS
4. Implementation and consolidation of a multicluster EDGE/CLOUD infrastructure for DELIS by using K3S and FLUIDOS/LIQO
5. Containerisation of all DELIS services
6. YAML Deployments of all DELIS services into the new infrastructure
7. Peering between clusters and services Offloading by using also the Affinities Rules
8. Analysis of the DELIS health status through the use of the CheckMK tool which will be the base point for Fluidomos analysis and comparison
9. Test and validation process was by acting on the following tasks:
    i. Standard testing tasks: unit, integration, system (functional, non-functional, performance), and usability testing.
    ii. Validation: The validation was executed at the pilot site (The Vemar building) using predefined test cases. During the validation process, attention was given to optimizing the performance of the FLUIDOMOS system. This may include identifying and resolving any performance issues, as well as testing the system's scalability to ensure it can handle heavier operational loads in the future.
10. Reporting critical issues encountered during FLUIDOS adoption for the FLUIDOMOS project

© 2022-2025 FLUIDOS

Funded by Horizon Europe Framework Programme of the European Union

WorkPlan, Milestones and KPIs

All previously listed results are carried out in the form of outcomes produced during the work activities we planned in the submission of the FLUIDOMOS proposal.

They are the following:

Work Plan Activity Name	Activity description	Plan Duration	Outcome
FLUIDOS analysis for the modeling of FLUIDOMOS	Study of Fluidos environment and application of its principles/features to the modeling of FLUIDOMOS architecture. The analysis of the nature and complexity of the architectural model will be completed as part of this activity. The key elements that make up the model will be described, such as the hardware and software components, the relationships between them, and how they contribute to the overall structure of the system. It will be illustrated how the architectural model meets the requirements and objectives of the project.	[m1-m2]	Technical report of the FLUIDOMOS model
Microservices Design, Redefinition, and Integration	Build images of microservices. All DELIS services will be ported into software services in the format of redistributable images that will be used to containerize services to expose as K8S pods into workers' nodes.	[m2-m6]	Software images of FLUIDOMOS microservices interfacing IOT
Coding the infrastructure	Coding the FLUIDOMOS infrastructure and microservices: Coding activities to produce the YAML files needed to implement all FLUIDOMOS entities in K8s formats. Designing and developing the intent-driven approach of FLUIDOS applicable to FLUIDOMOS	[m4-m7]	A comprehensive set of YAML files for the implementation of FLUIDOMOS
Testing and Validation	This activity will involve standard testing tasks: unit, integration, system (functional, non-functional, performance), and usability testing. The validation will be executed at the pilot site (The Vemar building) using predefined test cases. During the validation process, attention will be given to optimizing the performance of the FLUIDOMOS system. This may include identifying and resolving any performance issues, as well as testing the system's scalability to ensure it can handle heavier operational loads in the future.	[m5-m8]	Use case test report and validation of FLUIDOMOS FLUIDOMOS Setup and User guide

Table 1: Outcomes

© 2022-2025 FLUIDOS

Funded by Horizon Europe Framework Programme of the European Union

All activities results were also organized with the following milestones:

Milestone name	Milestone Description	Delivery Month
M1 - FLUIDOMOS model completed	The architectural model of FLUIDOMOS has been realised.	m2
M2 - Infrastructure and Microservices of FLUIDOMOS developed	FLUIDOMOS fundamental services and infrastructure are up and can be used for defining and implementing custom domotic systems	m7
M3 - FLUIDOMOS tested and validated	FLUIDOMOS has been implemented, tested, and validated at the pilot site.	m8

Table 2: Project Milestones

And, finally we also have defined and verified , for each milestones, the following KPIs to measure achievement of results:

M1 - The FLUIDOMOS architectural model is completed. It will be the basis for the completion of the next project phases.

KPI 1.1. SCRUM templates for planning and tracking project activities were defined and used. Tools for communication among project participants and sharing of project documentation are configured and used.
KPI 1.2. The Architectural Model is defined and the technical report on the FLUIDOMOS model is complete.
KPI 1.3. There is a census of the services to be migrated. Each service has an identification number, a signature, and a description of its behavior.
KPI 1.4. Accounting records and documentation for expenditure reporting have been prepared and maintained.

M2 - Fundamental microservices provided by FLUIDOMOS which are available and operational.

A review of the structure of the FLUIDOMOS infrastructure and how it has been configured to support microservices will be produced, including information on the underlying technologies used, the scalability of the infrastructure, and its ability to manage an increasing number of users and devices.
We assess that the fundamental microservices of DELIS are 50% of the global number of services to migrate. Accessory services (FUIDOMOS usage for defining and implementing a pilot site) will be available at the milestone M3.

Funded by Horizon Europe Framework Programme of the European Union

KPI 2.1. The fundamental services are available in FLUIDOMOS (50% of the global number) in the format of K8S pod
KPI 2.2 Container metrics will provide insights into the behavior and performance of individual containers running in a Kubernetes pod:

- CPU usage, for measuring the amount of CPU resources used by a container
- Memory utilization, for tracking the amount of memory used by a container
- Network utilization, for measuring the amount of network traffic sent and received by a container

KPI 2.3 Microservices metrics will provide insights into the behavior and performance of microservices in a Kubernetes pod:

- Service availability, for measuring the percentage of time the service is available
- Service health, for tracking the service status and response time
- Service performance, for measuring the throughput and latency of the service

M3 - FLUIDOMOS tested and validated.

The pilot site validation will indicate that FLUIDOMOS has been implemented and tested in a real operational environment. Validation on this pilot site will be essential to compare the system's expected behavior with its actual execution in real conditions.
The results obtained during the validation phase will be shared, including the performance of the system and the resolution of any problems detected. The produced documentation will indicate how users can customize and configure their home automation systems using FLUIDOMOS. This may involve defining automation rules, configuring devices, integrating new devices, and more.

KPI 3.1. Accessory services for the implementation of the pilot site using FLUIDOMOS are implemented .
KPI 3.2. The produced documentation reports the technical analysis of performances and scalability.
KPI 3.3. User Feedback Modules have been collected and analyzed.
KPI 3.4. The user guide is completed
KPI 3.5 Accounting records and documentation for expenditure reporting have been prepared and maintained for all the project activities.

In the next paragraphs, we expose all the milestones and KPI details.
Note that KPIs from 1.1 to 2.1 have been developed and exposed in the deliverable D 1.2 (and will be reported in this deliverable anyway). KPI 2.2 and KPI 2.3, partially produced in D1.2., have been completed.

Funded by Horizon Europe Framework Programme of the European Union

Milestone M1

The FLUIDOMOS architectural model is completed. It will be the basis for the completion of the next project phase.

KPI 1.1

SCRUM templates for planning and tracking project activities were defined and used. Tools for communication among project participants and sharing of project documentation are configured and used.

The configuration of the software Odoo used for project management considered 4 project views, each with different working methods, tools, and purposes:

1) Administrative view. Contains all the tasks required to fulfil administrative tasks related to the project.
2) Product view. It contains the tasks related to developing high-level functional and non-functional specifications planned for the migration of DELIS into FLUIDOMOS.
3) Knowledge management view. Tasks of this view have been necessary for the in-depth study of the basic technologies (Docker, Kubernetes, Liqo, Fluidos) plus the further knowledge acquisition tasks that will arise as the project progresses.
4) Communication view. Communication activities are divided into internal communication with the working group and external communication with FLUIDOS representatives.

The following figure shows the implementation of these views at the top level.

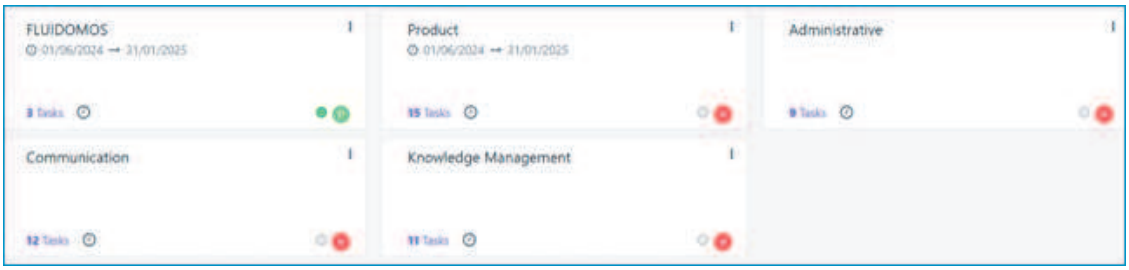
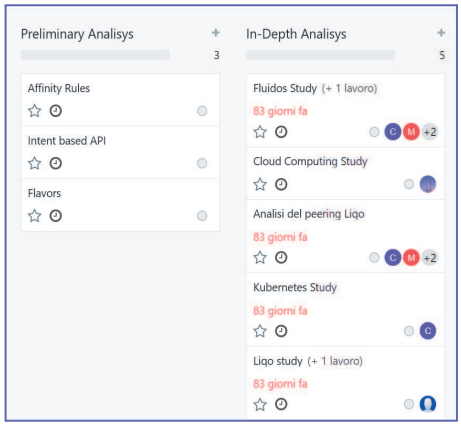


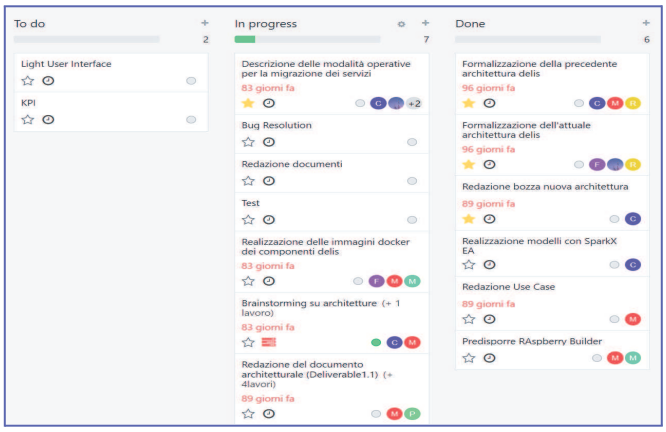
Figure 3: FLUIDOMOS project views by using Odoo

The templates for planning and tracking project activities are based on kanban cards as shown for:

- The tasks for knowledge acquisition
- The monitoring of project tasks according to the SCRUM methodology used for development.

Funded by Horizon Europe Framework Programme of the European Union





a) Knowledge management tasks b) Development tasks.

Figure 4: FLUIDOMOS project tasks

KPI 1.2

The Architectural Model is defined and the technical report on the FLUIDOMOS model is complete.

The Fluidomos Architecture describes how to migrate the Delis Domotic Building Cloud Management Platform into microservices runnable across a multi-clustered environment improved by Fluidos.

In-depth, it provides a structural representation of the following:

A classification for the new containerized services provided by Fluidomos

This section of the architecture diagram (FIG XX) describes a basic classification of DELIS software components and how to migrate them into containerized microservices or applications to comply with the FLUIDOS Computing Continuum operations. It also gives structural guidelines for building software images characterizing Kubernetes deployments.

The multi-clusters topology

This defines the clusters' topology for implementing Computing Continuum serving Edge controllers. It also defines peering, consumers, and providers of resources when services offloading events occur.

Services deployment guidelines

Funded by Horizon Europe Framework Programme of the European Union

Defines which and where services are initially deployed inside the FLUIDOS multi-cluster boundaries.

Service offloading guidelines

Define which and where services can be offloaded across the clusters using FLUIDOS.

We start by representing the current DELIS system architecture followed by the new Fluidomos design rethink regarding FLUIDOS and LIQO technologies.

Delis Architecture

In the current section, we show a diagram to give a brief representation of DELIS architecture followed by a simple description of the DELIS system and its main software components:

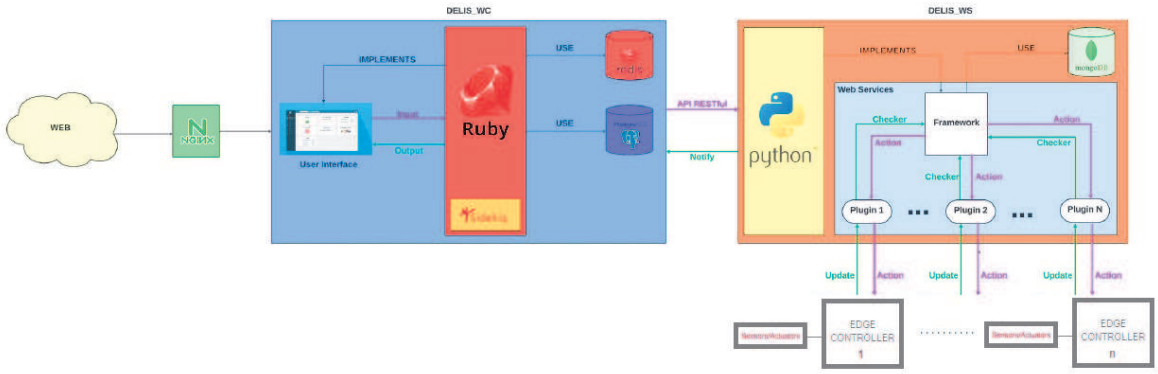


Figure 5: The current DELIS architecture

Briefly, DELIS is a Cloud Platform designed to build, manage, and maintain domotic systems. In this scenario, it offers a web dashboard to execute all the commonly used actions for configuring domotic devices to be managed, acting on them, monitoring the state, and so on.

DELIS software components shown in the previous diagram are:

1) User interface components to provide the Dashboard web application. They are:
    a) NGNIX HTTP server to run and provide web access to the DELIS Dashboard
    b) DELIS dashboard itself, a web application built using the Ruby on Rails framework
    c) SideKiq (used by the Dashboard), a library for efficiently managing background jobs on Ruby

Funded by Horizon Europe Framework Programme of the European Union

d) REDIS and PostgreSQL, the application in memory and storage databases

2) Web Service, for interfacing sensors and actuators managed by the Edge controllers: DELIS provides different web services for interfacing with different branded technologies such as SonOFF, Zway, etc.. In-depth, all web services are developed using Python and a custom framework developed by Vemar. The framework is the result of Vemar's research in the domotic system. It gives all primitives and abstractions representing how to interface and interact with the domotic devices. By using the framework developers can easily write Custom Plugins interfacing with different branded software and protocols at the Edge level. Mongo DB is used to store sensors/actuator data

3) Hardware components, like sensors, and actuators directly connected to edge controllers, ones equipped with branded software to communicate with them.

The Fluidomos Architecture: Delis migration on Fluidos

The new FLUIDOMOS architecture, which is derived from the previous regarding DELIS, is shown in the following diagram. In short:

- It redesigns software components in terms of Kubernetes deployments

- It redesigns the execution environment in terms of Kubernetes clusters, powered by FLUIDOS and LIQO

- It organizes the hardware equipment under new boundaries views focused on the physical location of the domotic units and buildings

Funded by Horizon Europe Framework Programme of the European Union

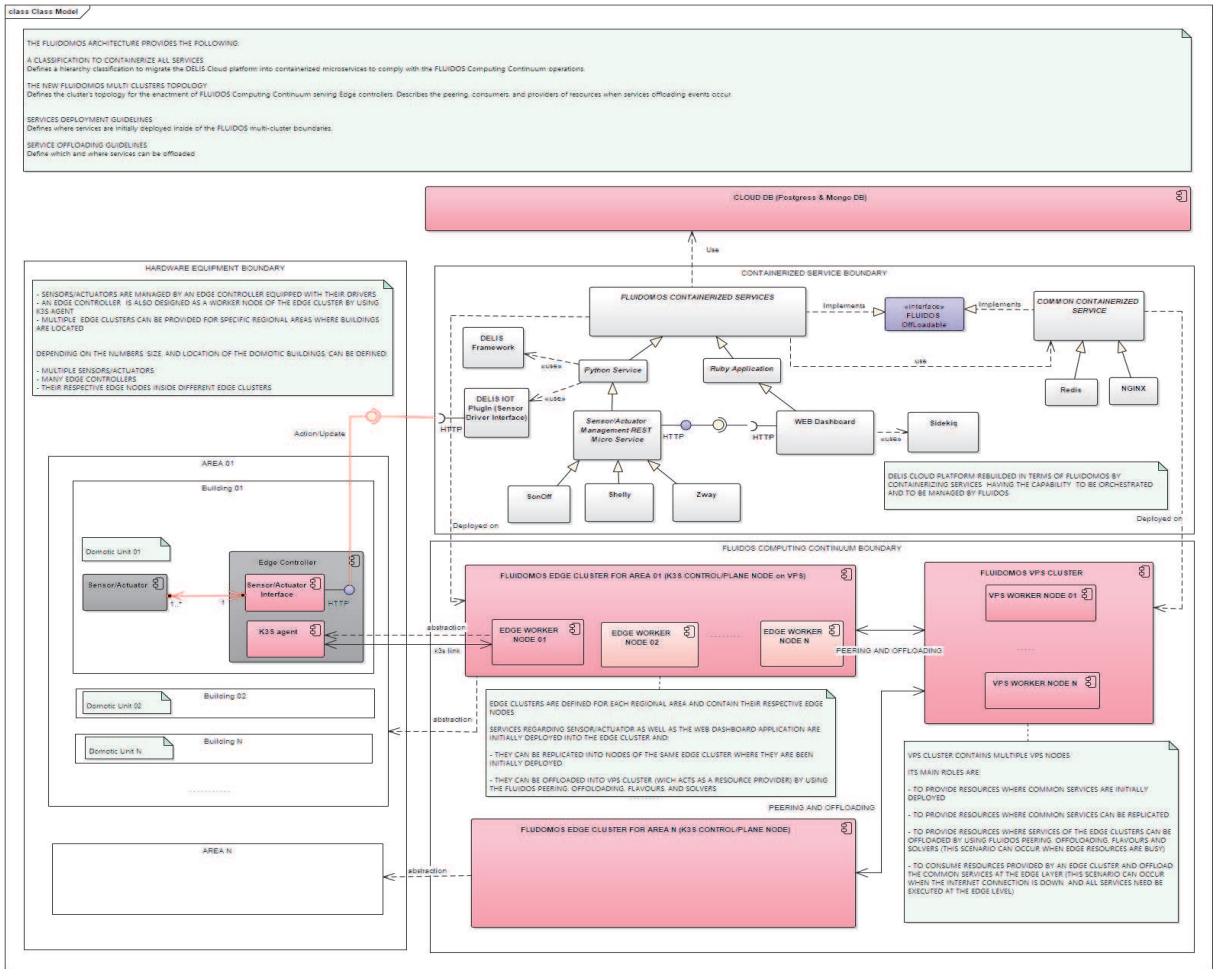


Figure 6 The FLUIDOMOS architecture

The architecture diagram can be evaluated from three separate points of view, identifying three different boundaries: Hardware, Services, and the Fluidos Computing Continuum. Each boundary represents an isolated level involving various technologies and skills.

HTTP interfaces allow communication among different DELIS services translated into Kubernetes deployments. Thanks to Fluoidos, service execution and orchestration are spanned over Edge and Cloud resources through peering and offloading functionalities.

The Hardware Boundary

The first, shown on the left, concerns the hardware equipment. Both hardware equipment with its related branded software are substantially unchanged compared to the previous one by Delis. This boundary describes how typically an edge controller manages sensors and actuators of a home automation system to define the domotic unit to manage.

Funded by Horizon Europe Framework Programme of the European Union

In the new Fluidomos vision, many domotic units can be realized for different places or buildings, which are in turn organized in different geographical areas. As shown using "abstraction" arrows, different areas are respectively managed, by edge clusters associated with them. A single edge cluster is itself composed of edge nodes representing the edge controller inside domotic units located in a geographical area.

Services Boundary

It represents a basic model to facilitate the migration of Delis software components into service must be orchestrated and deployed in Kubernetes clusters. The new Fluidomos services should be:

- Off-loadable by Fluidos into the provider node
- Deployable into the Fluidomos clusters

To facilitate the developments, services are organized and composed by using the following structure:

- Fluidomos Services Images
    - Python Web Services for the different brands. They are software images built using the following Delis components:
        - DELIS Framework (the same one described before under Delis architecture.)
        - Custom Plugin to interface branded protocols
    - The Dashboard application image. The same one described before under Delis architecture (but increased with a few features described later)
    - Common Image: Are images of common open source software (REDIS, MongoDB etc.)

All the software images are exposed as Kubernetes deployments. They are initially deployed in the cluster and/or can be offloaded across clusters as shown in the figure

Fluidos Computing Continuum Boundary

Placed on the bottom-right of the figure, it represents the new computing abstract environment based on a multi-cluster topology that can be orchestrated and managed by Fluidos. The abstract environment is mapped on many K3s clusters: one for each Area defined in the hardware boundaries. The architecture identifies this type of k3s cluster with the name of Edge Cluster because each node is defined using a k3s agent running inside the Edge controller. Moreover, a specialized cluster, called VPS Cluster, is designed to have multiple VPS nodes (each k3s agent runs on a cloud virtual private server) and is used to provide resources where offload Edge Clusters' services.

© 2022-2025 FLUIDOS

Funded by Horizon Europe Framework Programme of the European Union

In brief:

- Edge clusters are defined for each Regional Area and contain their respective Ege Nodes
- Services regarding sensor/actuator as well as the web dashboard application are initially deployed into the Edge Cluster and:
    - They can be replicated into nodes of the same Edge Cluster where they are been initially deployed
    - They can be offloaded into the VPS cluster (which acts as a resource provider) by using the Fluidos peering, off-loading, flavors, and solvers.

Cloud DB

The persistent storage engines, used by Fluidomos services, are provided by external Cloud Services. Currently, this choice is due to the complexity of managing stateful applications in Kubernetes environments. However, the new Fluidomos web Dashboard, using the edge file system, will manage the persistence of a minimum set of data necessary to always guarantee the operation of the system (but with reduced functionalities). So that, even in case of unavailability of Internet access (determining the unavailability of the Cloud DB) the overall service, running at the edge level, will be available for users.

Moreover, the Postgres Physical Replication between Cloud and Edge nodes is actually under evaluation to guarantee data are always accessible at the edge level even when Cloud access is unavailable. Physical replication of data will only involve data partitions, designed for each edge node so that each edge node will have its data at the local level. Since this solution is currently under development, its representation in the architecture diagram has been intentionally omitted.

The Vemar Team also includes evaluating the use of Longhorn storage at the Kubernetes level in the future.

Implementation

The implementation of the FLUIDOMOS architecture was developed through several phases in which, gradually, we started by creating a k3s cluster acting as a FLUIDOS consumer, then by adding a cluster with a FLUIDOS provider role, and, finally, by adding more worker nodes on both. Moreover, all the technical details of these steps are specified in the following files:

Filename	Description
Fluidomos_Installation_Procedure_Step-01.docx	The baseline provided by Fluidos Mentor on which was started the study.

Funded by Horizon Europe Framework Programme of the European Union

Fluidomos_Installation_Procedure_Step-02.docx	The first attempt to install a clustered environment. Created an environment with two clusters with only 2 control planes according to the previous installation procedure.
Fluidomos_Installation_Procedure_Step-03.docx	Second attempt to install a clustered environment. Created an environment with 2 clusters, each with 1 control plane and 1 worker node. Moreover, a Rancher monitoring system was installed. https://fluidomos.vemarsas.it
Fluidomos_Installation_Procedure_Step-04.docx	A failed attempt to modify the provider resources into the multi cluster environment having 2 control plane
Fluidomos_Installation_Procedure_Step-05.docx	A successful attempt to modify the provider resources into the multi cluster environment having 2 control planes.
Fluidomos_Installation_Procedure_Step-06.docx	The new clustered environment is now in Public Datacenters. It was created using 2 clusters, each with 1 control plane and 1 worker node. Providers' resources are now changed according to the Delis requirements https://fluidomosv2.vemarsas.it
Fluidomos_Installation_Procedure_Step-07.docx	Added Edge node configuration information
FLUIDOMOS Guide.md	The overall procedure to implement the FLUIDOMOS infrastructure of the test case to manage the home automation system at Vemar Sas has been finalized

Table 3: FLUIDOMOS Installation steps

The project's public repository is now available at:
https://github.com/vemarsas/FLUIDOMOS

To enact the DELIS platform using FLUIDOS, you can access and consult the cluster infrastructure implementation guide with all related YAML distributions.
In the diagram below there is a diagram representing the FLUIDOMOS use case cluster topology to realize the domotic building of Vemar Sas.

Funded by Horizon Europe Framework Programme of the European Union

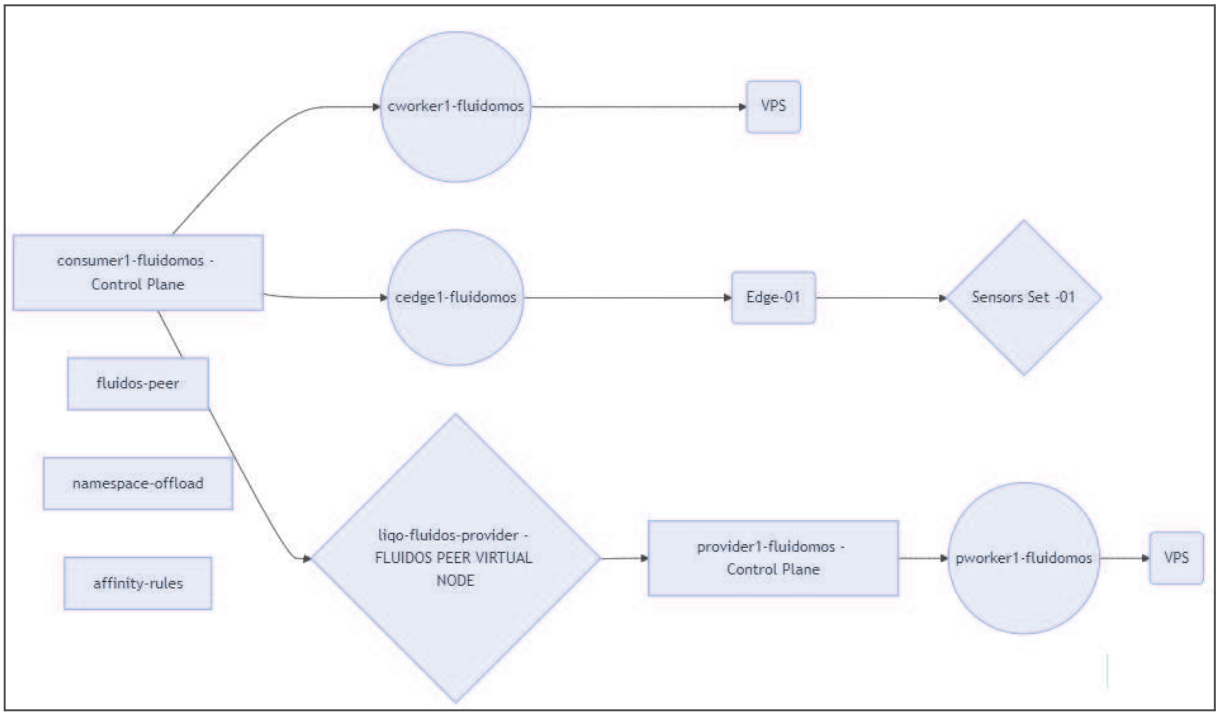


Figure 7 - Clusters topology of FLUIDOMOS Use Case

KPI 1.3

There is a census of the services to be migrated. Each service has an identification number, a signature, and a description of its behavior

Delis architecture, we just presented in the previous paragraph, is shown again in the following figure, and it was used to identify the main services to be migrated:

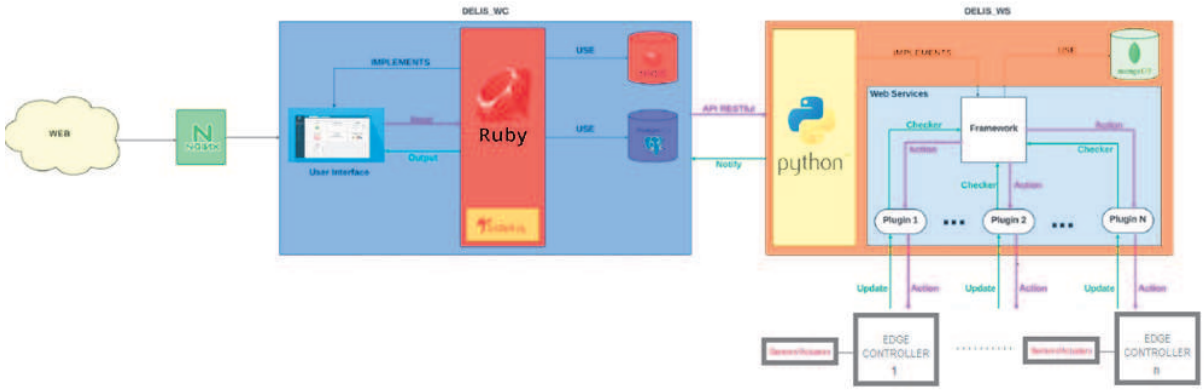


Figure 8: Delis Architecture

According to this, the main Delis services and plugin components migrated, in the form of microservices for the FLUIDOS seamless computing environment, are the following:

1. delis_wc: Is the main Delis Web user interface & Core

Funded by Horizon Europe Framework Programme of the European Union

2. delis_sidekiq: A library for efficiently managing background jobs of the Delis Web Application
3. delis_websocket: A library for efficiently managing web socket communication in the Delis Web Application. It doesn't appear in the figure above because it is a Core Component.
4. delis_ws: This is a Framework developed by Vemar for writing plugins and developing web services
5. zway plugin: This is a plugin for interacting with Z-wave devices
6. sonoff plugin: This is a plugin for interacting with Sonoff devices
7. shelly plugin: This is a plugin for interacting with Shelly devices (under test)
8. Chromecast: A multimedia manager (under development)

Services and plugins from 1 to 6 are currently deployed into the FLUIDOMOS environment (https://fluidomosv2.vemarsas.it). Plugins and 8 is at the moment under development and not still running in FLUIDOMOS.

The Delis services migration process's technical report is itself a project outcome. It organizes the migration process into four steps:

- Creating A Docker Image
- Creating A Multi-platform Docker Image
- Migrating To Kubernetes
- Building Vemar sas Registry

All technical details about these steps are described in FLUIDOMOS repository at https://github.com/vemarsas/FLUIDOMOS/ file:

DELIS Service Migration Process.md

The Endpoints/APIs signature of the main Delis Services/Components are reported in the following tables. They are usable to better understand the technical details, constraints, and dependencies of Delis services to prevent issues in the migration process.

Funded by Horizon Europe Framework Programme of the European Union

N	SERVICES	DESCRIPTION	REQUIREMENTS	STATUS	NOTE
1	delis_wc	delis Web & Core	2vCPU 512 MB RAM Internet Connection >= 1 Gbps	Migrated	Installed and tested
2	delis_sidekiq	Service for Alerts and signalations external communication	2vCPU 512MB RAM	Migrated	Installed and tested
3	delis_websocket (cable)	websocket sercvice for client browser and app communication	2vCPU 512MB RAM	Migrated	Installed and tested
4	delis_ws	framework for plugin interaction end web service development	Environment variables: - PYDELIS_DEBUG: false - PYDELIS_SECRET_KEY: 16 byte random exadecimal string - PYDELIS_API_KEY: 16 byte random exadecimal string - PYDELIS_DB: MongoDB database name - PYDELIS_HOST: IP or DNS of the DBMS host - PYDELIS_PORT: MongoDB database port - PYDELIS_USERNAME: DBMS user name - PYDELIS_PASSWORD: DBMS user password	Migrated	Installed and tested
5	pydelis_zway	plugin for z-wave device integration	1 vCPU 800 MB 256 MB RAM	Migrated	Installed and tested
6	pydelis_sonoff	plugin for sonoff device integration	1 vCPU 800 MB 256 MB RAM	Migrated	Under testing
7	pydelis_shelly	plugin for shelly device integration	1 vCPU 800 MB 256 MB RAM	Migrated	Under testing
8	pydelis_chromecast	plugin to intercat with multimedia chromecast environment		To Migrate	Under development
9	pydelis_onvif	plugin to interact with IP cameras that support the ONVIF standard		To Migrate	Under development
Other features will be added to delis_wc and delis_ws. They are:					
	scenemgmt	to manage status update	Dedicated database	Done	Integrated in delis_wc
	test	To execurte interfacing tests	In evaluation	To be done	To be integrated in delis_ws

Table 4: Delis Services And Plugin

Funded by Horizon Europe Framework Programme of the European Union

SERVICE/COMPONENT	ID	SIGNATURE	DESCRIPTION
delis_ws	run	def run(cls, version, infomodule:tuple=None)	Runs the pydelis framework
Runs the framework instance based on the version specified as a parameter. In more detail, the specific plugin module for a technology is traced and configured and the corresponding web service is started through the use of Flask and Waitress.			
delis_ws	/<ws_name>	def post(self)	RESTful API for creating the web service via HTTP POST call.
Implements the web service creation logic. Check the token present in the request header and if there are no errors, create the web service by saving it in the database. The response in JSON format contains the status code 200, name and endpoint for HTTP calls to the web service. If the token is invalid, a 401 status code and message: 'Unauthorized access' is returned.			
delis_ws	/<ws_name>	def get(self)	RESTful API for displaying the web service via HTTP GET call.
Check the token and if there are no errors, returns the status code 200, the name and endpoint of the web service if present. Otherwise status code 404 with message: 'Web service not found'.If the token is invalid, a 401 status code and message: 'Unauthorized access' is returned.			
delis_ws	/<ws_name>	def delete(self)	RESTful API for delete the web service via HTTP DELETE call.
Implements web service deletion logic. Check the token and if there are no errors delete the web service from the database. The response in JSON format contains the status code 200 and the message: 'Web service successfully deleted'. If the web service is not present, the status code 404 and the message: 'Web service not found' are returned. If the token is invalid, a 401 status code and a message: "Access not authorized" are returned.			
delis_ws	<endpoint>/<ws_name>/installation	def post(self)	RESTful API for creating an installation via HTTP POST call.
Implements the installation creation logic. It checks the token present in the request header, the name of the installation is passed as a parameter and if there are no errors it creates the installation saving it in the database. The response in JSON format contains the 200 status code, id, name and token for HTTP calls to be made to this installation. If the token is invalid, a 401 status code and a message: "Access not authorized" is returned.			
delis_ws	<endpoint>/<ws_name>/installation/<installation_id>	def get(self, installation_id)	RESTful API for viewing installation via HTTP GET call.
It checks the token and if there are no errors it returns the status code 200 and the name of the installation if present. Otherwise status code 404 with message: 'Installation not found' If the token is invalid, a 401 status code and a message: "Access not authorized" is returned.			
delis_ws	<endpoint>/<ws_name>/installation/<installation_id>	def patch(self, installation_id)	RESTful API to modify installation via HTTP PATCH call.
Implements the installation editing logic. It checks the token present in the request header, the new name of the installation is passed as a parameter and if there are no errors modify the installation saving it in the database. The response in JSON format contains the 200 status code with message: 'Installation name updated'. Otherwise status code 404 with message: 'Installation not found' If the token is invalid, a 401 status code and a message: "Access not authorized" is returned.			
delis_ws	<endpoint>/<ws_name>/installation/<installation_id>	def delete(self, installation_id)	RESTful API for delete the installation via HTTP DELETE call.
Implements the installation deletion logic. It checks the token present in the request header,if there are no errors delete the installation from the database. The response in JSON format contains the 200 status code with message: 'Installation successfully deleted'. Otherwise status code 404 with message: 'Installation not found' If the token is invalid, a 401 status code and a message: "Access not authorized" is returned.			
delis_ws	<endpoint>/<ws_name>/<installation_id>/der	def post(self, installation_id)	RESTful API for creating a DER via HTTP POST call.
Implements the DER creation logic. It checks the token present in the request header and name, ip, port, protocol, username and password of the DER are passed as parameters and if there are no errors it creates the DER saving it in the database. The response in JSON format contains the 200 status code, id, token for HTTP calls to be made to this DER and the list of DEDs. If the token is invalid, a 401 status code and a message: "Access not authorized" is returned. If the installation_id does not match any current installation, a 404 status code and a message: 'Installation not found' is returned. If there is an error in the definition of a DER within the plugin, the status code 500 is returned with the message: 'Error in plugin definition'.			
delis_ws	<endpoint>/<ws_name>/der/<der_id>	def get(self, der_id)	RESTful API to view the DER via HTTP GET call.
Check the token present in the request header and the name, IP, port, protocol, username and if there are no errors it shows the DER if present in the database. The response in JSON format contains the status code 200, the name and uri of the DER and the list of DEDs. If the token is invalid, a 401 status code and a message: "Access not authorized" is returned. If the der_id does not match any current DER, a 404 status code and a message: "Der not found" is returned.			
delis_ws	<endpoint>/<ws_name>/der/<der_id>	def delete(self, der_id)	RESTful API to delete DER via HTTP DELETE call.
Implement DER deletion logic. Check the token present in the request header and if there are no errors the DER is deleted from the database. The response in JSON format contains the status code 200 and the message: "Der deleted successfully". If the token is invalid, a 401 status code and a message: "Access not authorized" is returned. If the der_id does not match any current DER, a 404 status code and a message: "Der not found" is returned.			
delis_ws	<endpoint>/<ws_name>/der/<der_id>/<field>	def patch(self, der_id, field)	RESTful API to modify DER via HTTP PATCH call.

Funded by Horizon Europe Framework Programme of the European Union

Implement the DER modification logic. Check the token present in the header of the request, the name field can be modified by passing the new name as parameters, the network field by passing the new values for IP, port and protocol as parameters, the credentials field by passing the new values for username as parameters and password. If there are no errors, modify the Der by saving it in the database. The response in JSON format contains the status code 200 with the message: "Der updated successfully". If the token is invalid, a 401 status code and a message: "Access not authorized" is returned. If the der_id does not match any current DER, a 404 status code and a message: "Der not found" is returned. If there is an error within the plugin definition, the status code 500 is returned with the message: 'Error in plugin definition'. If the specified field is different from the 3 indicated, the response contains status code 500 with message: 'There is a generic error, please contact the system administrator'.			
delis_ws	<endpoint>/<ws_name>/der/<der_id>/ded	def post(self, der_id)	RESTful API to create all DEDs related to the DER via HTTP POST call.
Implements the DEDs creation logic. Check the token present in the header of the request. If there are no errors, it creates the DEDs relating to the DER specified within the plugin code and saves them in the database. The response in JSON format contains the status code 200 and the list of id of each DED. If the token is invalid, a 401 status code and a message: "Access not authorized" is returned. If the der_id does not match any current DER, a 404 status code and a message: "Der not found" is returned. If there is an error within the plugin definition, the status code 500 is returned with the message: 'Error in plugin definition'.			
delis_ws	<endpoint>/<ws_name>/der/<der_id>/ded/<ded_id>	def post(self, der_id, ded_id)	RESTful API to create a DED related to the DER via HTTP POST call.
Implements the DED creation logic. Check the token present in the header of the request. If there are no errors, it creates the DED relating to the DER specified within the plugin code and saves it in the database. The response in JSON format contains the status code 200 and the DED id. If the token is invalid, a 401 status code and a message: "Access not authorized" is returned. If the der_id does not match any current DER, a 404 status code and a message: "Der not found" is returned. If there is an error within the plugin definition, the status code 500 is returned with the message: 'Error in plugin definition'. If the entered DED id is already present, status code 500 is returned with the message: 'An existing device with the provided ID already exists'. If among the DEDs physically added to the DER there is no DED with the entered ID, a 404 status code and message are returned: 'Ded not found'.			
delis_ws	<endpoint>/<ws_name>/der/<der_id>/ded/<ded_id>	def get(self, der_id, ded_id)	RESTful API for displaying the DED via HTTP GET call.
Check the token present in the header of the request. If there are no errors, it shows the DED present in the database with that id. The response in JSON format contains the status code 200 and the DED with actions, information, properties and widget. If the token is invalid, a 401 status code and a message: "Access not authorized" is returned. If the der_id does not match any current DER, a 404 status code and a message: "Der not found" is returned. If the ded_id does not match any current DED, a 404 status code and a message: "Ded not found" is returned.			
delis_ws	<endpoint>/<ws_name>/der/<der_id>/ded/<ded_id>	def delete(self, der_id, ded_id)	RESTful API to delete DED via HTTP DELETE call.
Implement DED deletion logic. Check the token present in the header of the request. If there are no errors, delete the DED present in the database with that id. The response in JSON format contains the status code 200 and a message:. If the token is invalid, a 401 status code and a message: "Access not authorized" is returned. If the der_id does not match any current DER, a 404 status code and a message: "Der not found" is returned. If the ded_id does not match any current DED, a 404 status code and a message: "Ded not found" is returned.			
delis_ws	<endpoint>/<ws_name>/der/<der_id>/ded/<ded_id>/<field>	def patch(self, ded_id, ded_id, field)	RESTful API to modify DED via HTTP PATCH call.
Implement the DED modification logic. Check the token present in the header of the request, the information field can be modified by passing the new information as parameters, the actions field by passing the new actions as parameters, the properties field by passing the new properties as parameters and the widget field by passing the new widget as parameters. If there are no errors, modify the DED by saving it in the database. The response in JSON format contains the status code 200 with the message: "Ded updated successfully". If the token is invalid, a 401 status code and a message: "Access not authorized" is returned. If the der_id does not match any current DER, a 404 status code and a message: "Der not found" is returned. If the ded_id does not match any current DED, a 404 status code and a message: "Ded not found" is returned. If the specified field is different from the 4 indicated, the response contains status code 500 with message: 'There is a generic error, please contact the system administrator'.			
delis_ws	<endpoint>/<ws_name>/der/<der_id>/ded/<ded_id>/action/<action>	def post(self, ded_id, der_id, action)	RESTful API to perform the DED action via HTTP POST call
Implements the action execution logic of a DED. Check the token present in the header of the request and the input value is passed as a parameter. If there are no errors the action is executed. The response in JSON format contains the status code 200 with the message: "Action done successfully". If the token is invalid, a 401 status code and a message: "Access not authorized" is returned. If the der_id does not match any current DER, a 404 status code and a message: "Der not found" is returned. If the ded_id does not match any current DED, a 404 status code and a message: "Ded not found" is returned. If the specified action is not among those offered by DED the response contains status code 500 with message: 'There is a generic error, please contact the system administrator'.			
delis_ws	<endpoint>/<ws_name>/der/<der_id>/monitor	def get(self, der_id)	RESTful API to show DEDs whose properties have changed.
Check the token present in the header of the request. If there are no errors, the DED properties in the database are updated. The response in JSON format contains the status code 200 and the list of DEDs that have changed status. If the token is invalid, a 401 status code and a message: "Access not authorized" is returned. If the der_id does not match any current DER, a 404 status code and a message: "Der not found" is returned.			

Table 5: Delis WS API

Funded by Horizon Europe Framework Programme of the European Union

SERVICE COMPONENT	ID	SIGNATURE	DESCRIPTION
zway, sonoff, shelly, chromecast, onvif	main	def main()	Invokes execution of the delis_ws framework
The run() method of PyDelis is called and the string indicating the requested framework version is passed as a parameter.			
zway, sonoff, shelly, chromecast, onvif	__init__	def __init__(self, name, uri, username, password)	DER initialization method
Method to be necessarily implemented as it is inherited from the framework's DER abstract class. The name, uri, username and password parameters passed to the method when creating the DER instance of the plugin are initialized as properties.			
zway, sonoff, shelly, chromecast, onvif	name	def name(self)	Returns the value of the name of DER.
Method to be necessarily implemented as it is inherited from the framework's DER abstract class.			
zway, sonoff, shelly, chromecast, onvif	uri	def uri(self)	Returns the value of the uri of DER.
Method to be necessarily implemented as it is inherited from the framework's DER abstract class.			
zway, sonoff, shelly, chromecast, onvif	username	def username(self)	Returns the value of the username of DER.
Method to be necessarily implemented as it is inherited from the framework's DER abstract class.			
zway, sonoff, shelly, chromecast, onvif	password	def password(self)	Returns the value of the password of DER.
Method to be necessarily implemented as it is inherited from the framework's DER abstract class.			
zway, sonoff, shelly, chromecast, onvif	deds	def deds(self)	Returns the list of DEDs relating to the DER.

Funded by Horizon Europe Framework Programme of the European Union

Method to be necessarily implemented as it is inherited from the framework's DER abstract class. It encapsulates the methods for the acquisition, through HTTP calls, of the DEDs data relating to the DER, creating the objects corresponding to the different types of devices that may be present.			
zway, sonoff, shelly, chromecast, onvif	information	def information(self)	Returns the value of the information of DED.
Method to be necessarily implemented as it is inherited from the framework's DED abstract class. Returns a dict of DED data that does not change over time.			
zway, sonoff, shelly, chromecast, onvif	properties	def properties(self)	Returns the value of the properties of DED.
Method to be necessarily implemented as it is inherited from the framework's DED abstract class. Returns a dict of DED data that changes over time and represents the state of the device.			
zway, sonoff, shelly, chromecast, onvif	actions	def actions(self)	Returns the value of the actions of DED.
Method to be necessarily implemented as it is inherited from the framework's DED abstract class. Returns a list of strings indicating the actions that can be performed on the DED.			
zway, sonoff, shelly, chromecast, onvif	widget	def widget(self)	Returns the value of the widget of DED.
Method to be necessarily implemented as it is inherited from the framework's DED abstract class. Returns a dict with kays pre-established by the framework to specify to the delis_wc interface how to arrange the graphic components of the widget to represent the DED.			
zway, sonoff, shelly, chromecast, onvif	<action>	def <name_action>(self, uri, username, password, ded)	Performs a specific action for a DED.
It contains within it the logic to perform the desired action on the DED, through HTTP calls. It is mandatory that the method is called exactly like the corresponding string in the actions method list of the DED. This is a dictate of the delis_ws framework.			

Table 6: Delis Plugin API

KPI 1.4

Funded by Horizon Europe Framework Programme of the European Union

Accounting records and documentation for expenditure reporting have been prepared and maintained.

The partial documentation submitted in D1.2 has been finalized in the following table which reports all the project expenses from m1 to m2

Cost Category	Vemar sas	Costs	ICT Sistemi srl	Costs	Total amount
[a] Direct personnel cost	Marco Piscopia	[redacted]			
	Francesco Parrotta	3.510,00 €			
			Simone Nota	14.306,88 €	
			Alessandra Criscuolo	3.141,94 €	
					38.458,82 €
[b] Travel costs					
[c] Equipment costs			Sensors	1.490,00 €	1.490,00 €
[d] Other direct costs	Consulting by Rosario Carvello	14.000,00 €			
	Consulting by Mauro Salvia			15.915,43 €	
					29.915,43 €
Total		30.010,00 €		39.854,25 €	69.864,25 €

Table 7: Accounting records at D 1.2

In k3.5 we report a table showing the balance of the accounting records from m1 to m3

Milestone M2

Funded by Horizon Europe Framework Programme of the European Union

Fundamental microservices provided by FLUIDOMOS which are available and operational.

In the D1.2 we assess the fundamental microservices of DELIS were 50% of the global number of services to migrate. Accessory services (FUIDOMOS usage for defining and implementing a pilot site) are now available at the milestone M3.

KPI 2.1

The fundamental services are available

Below is the list of FLUIDOMOS services we provide:

YAML (on GitHub)	Type
deployment_cable.yaml	Deployment
deployment_pydelis_shelly.yaml	Deployment
deployment_pydelis_zway.yaml	Deployment
deployment_sidekiq.yaml	Deployment
deployment_wc.yaml	Deployment
namespace-fluidomos-datacenter.yaml	Deployment
namespace-fluidomos-edge.yaml	Deployment
namespace.yaml	Deployment
secret_regcred_fluidomos-system.yaml	
service_cable.yaml	Service
service_pydelis_shelly	
service_pydelis_zway.yaml	Service
service_sidekiq.yaml	Service
service_wc.yaml	Service

Table 8 : FLUIDOMOS Services

All YAML files are in available in the GIT repository:
https://github.com/vemarsas/FLUIDOMOS/tree/main/YAML

Funded by Horizon Europe Framework Programme of the European Union

KPI 2.2

Container metrics will provide insights into the behavior and performance of individual containers running in a Kubernetes pod

The objective of this activity and the following in KPI 2.3 is to conduct a performance analysis of Delis and its evolutionary implementation in Fluidomos to evaluate improvements or potential issues in terms of resource usage. To do this, the following key metrics have been monitored and compared.

METRICS

- CPU Load: The overall CPU load will be measured to verify the system's ability to handle running processes, both in the original version and the evolved version, and to identify changes in workload distribution.
- Memory Usage: Total memory usage will be monitored to assess how the system manages volatile memory (RAM) and whether the system's evolution leads to better allocation or management of available memory.
- Cache: The effectiveness of the cache memory will be analyzed, measuring how the system uses the cache to optimize read and write operations. Efficient cache usage can significantly improve system performance.
- Active & Inactive Memory: Active and inactive memory segments will be monitored. Active memory represents data and processes currently in use, while inactive memory includes unnecessary data that remains allocated but could be freed. Comparing the two system versions will allow us to identify improvements in memory management.
- RAM Used: The total RAM used in each system version will be recorded to identify whether the system's evolution introduces optimizations or increased resource demands.
- Uptime: System uptime will be recorded, i.e., the total continuous operation time without reboots. This will help us understand if there are improvements in the stability and reliability of the evolved system compared to the original one.

This monitoring intends to obtain a clear and objective view of the system's performance through concrete and comparable data between the two versions. The performance comparison will help us determine whether the system's evolution has led to significant improvements in terms of efficiency and resource management, or if new critical areas have emerged that need optimisation.
The monitoring results have been collected periodically and analyzed to highlight any differences in performance. This data guided us in making decisions regarding further optimizations or system changes.The monitoring results have been collected periodically and analyzed to highlight any differences in performance. This data guided us in making decisions regarding further optimizations or system changes.
The DELIS project is deployed on two distinct platforms: the stable one is DELIS while the development one, used for migration to FLUIDOMOS, is DELISDEV. Both systems operate on the same services, but while the first is intended for daily use by standard users, the second is dedicated to development. Preliminary analyses and activities for the migration to Fluidomos show signs of improvement on DELISDEV

Funded by Horizon Europe Framework Programme of the European Union

compared to DELIS, but the final comparison with Fluidomos will be decisive in assessing whether the interventions lead to a significant reduction in system load and an improvement in overall efficiency, especially in terms of CPU, memory management, and system stability.

DELIS MEASUREMENTS

A snapshot of the DELIS system taken at the beginning of September 2024 shows a clear picture of the original project's health status over a while since August 2023 to the time of the snapshot. Since then, all activities for the transition to the Fluidomos system have been carried out, and its final version has been in operation for a few days now. The next period will see the monitoring systems active for measurements on Fluidomos as well.

SYSTEM = DELIS
DATE = 05/09/2024
host = Local site Vemar
checkmk name = delis.vemarsas.lan

The following image shows a CPU load graph.

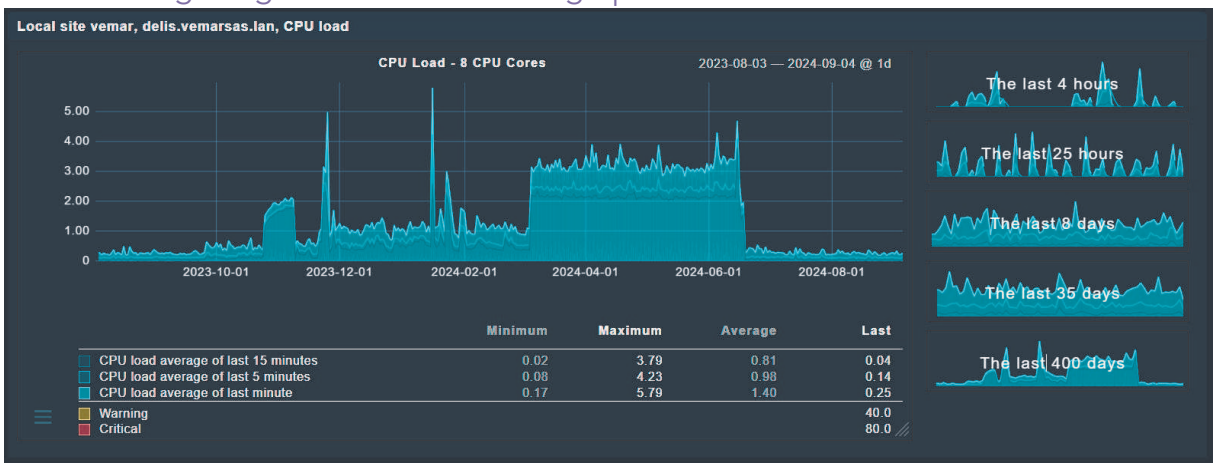


Figure 9 - DELIS CPU Load

The graph offers a detailed overview of CPU load over time.

- Load Trend: There is a wide variation in average loads, especially in the one-minute range (with a maximum of 5.79), suggesting possible periods of high activity or peaks of intensive use.
- Resource Management: Data shows how the CPU has been under variable load, which may indicate a need for optimization or load balancing.
- Stability: The absence of warning and critical lines suggests that the system has maintained stable operation without reaching dangerous load levels.

Monitoring CPU load is essential to ensure optimal performance and prevent bottlenecks. This data helps you identify critical moments and plan improvements. There seems to be ample room for intervention to optimize resources and reduce peak loads.

Funded by Horizon Europe Framework Programme of the European Union

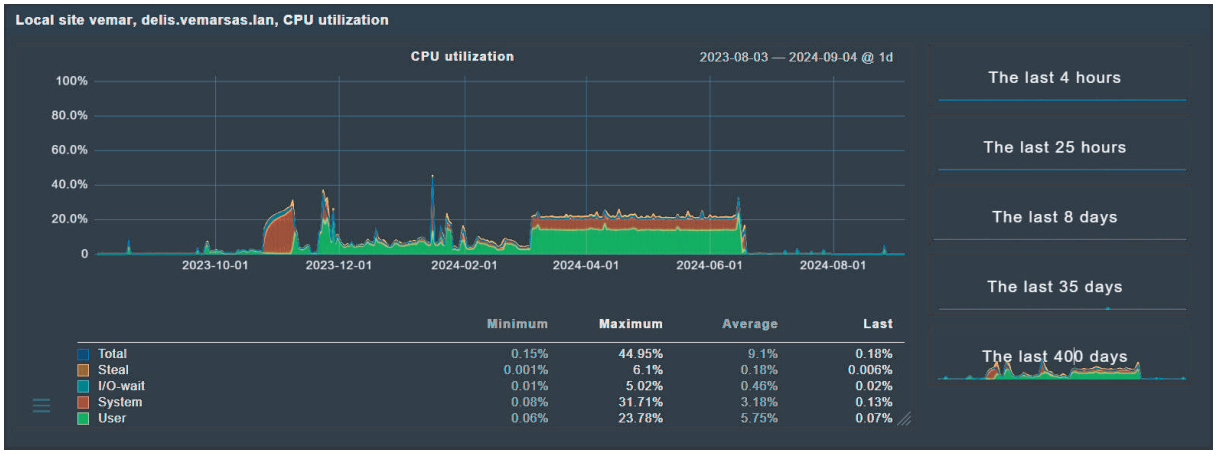


Figure 10 - DELIS CPU Utilization

The graph offers a comprehensive overview of CPU usage over time and details usage by showing different usage categories: Total, Steal, I/O wait, System, and User, each represented by different colours. The y-axis represents the percentage of CPU usage, which ranges from 0% to 100%, while the x-axis represents the timeline.

- Usage Trend: We observe that the total CPU usage has reached peaks of up to 44.95%, with an average value of 9.1%. These data suggest moments of high activity, possibly linked to intensive processes or periods of system stress.
- Usage Distribution: The CPU is mainly used for User (up to 23.78%) and System (up to 31.71%) processes, while I/O wait and Steal show significantly lower values, indicating less waiting for input/output operations. output and a low level of interference from virtual machines.
- Resource Management: The graph shows a fairly efficient distribution of CPU usage, with average values remaining under 10% for most categories, except for User and System processes which, together, make up the majority of the load.

Monitoring these metrics is crucial to optimizing system performance and identifying any bottlenecks. The data suggests that, while there are periods of high load, overall resource use appears to be under control. Continuing to monitor this data will help keep the system efficient and quickly identify any emerging issues.

© 2022-2025 FLUIDOS

Funded by Horizon Europe Framework Programme of the European Union

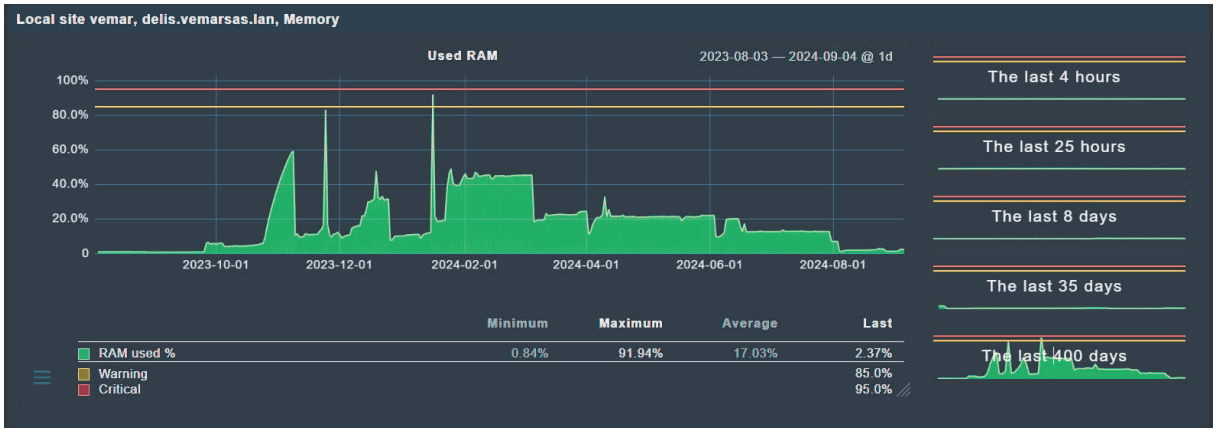


Figure 11 - DELIS Used RAM

The graph represents RAM usage and includes lines indicating the warning (85.0%) and critical (95.0%) thresholds. RAM usage is represented in green, with some significant spikes, reaching a maximum of 91.94%. The minimum RAM usage is 0.84% and the average is 17.03% while the last usage shown is 2.37%.

- Usage Trends: Significant peaks are observed during the months of December 2023 and February 2024, suggesting periods of increased activity or intensive workloads.
- Usage Distribution: The average RAM usage is 17.03%, with most of the time usage remaining below the warning and critical thresholds, indicating relatively efficient memory management.
- Resource Management: Warning and critical thresholds provide a good reference for monitoring when the system may need optimization or intervention to avoid overloads.

Monitoring RAM usage is crucial to ensuring optimal system performance and preventing bottlenecks. This data suggests that, while there are periods of high usage, overall resource use appears to be under control. Continuing to monitor and analyze this data will help keep the system efficient and quickly identify any emerging issues.

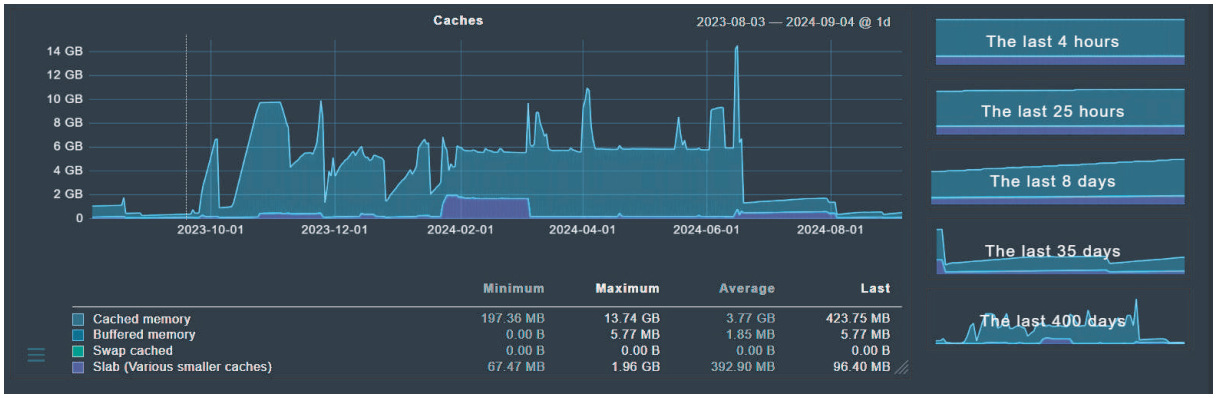


Figure 12 - DELIS Caches

The image represents the use of memory caches. Shows the use of different types of memory caches: Cached memory, Buffered memory, Swap cached, and Slab (various

Funded by Horizon Europe Framework Programme of the European Union

smaller caches). The y-axis represents the memory size in gigabytes (GB), while the x-axis represents the timeline.

- Usage Trend: We observe that the Cached memory has reached a maximum of 13.74 GB, while the average is 3.77 GB. Buffered and Swap cached memory show much lower values, with a maximum peak of 5.77 MB and 0 B respectively, indicating little to no usage.
- Usage Distribution: Slab memory has a maximum of 1.96 GB, with an average of 392.90 MB. This type of memory includes several smaller caches that are used for specific system operations.
- Resource Management: Overall memory cache usage appears to be under control, with most categories showing moderate average usage and manageable peaks.

Monitoring the use of memory caches is essential to ensure optimal system performance and prevent bottlenecks. The data suggests that while there are periods of high usage for cached memory, overall use of cache resources appears to be under control. Continuing to monitor this data will help keep the system efficient and quickly identify any emerging issues.

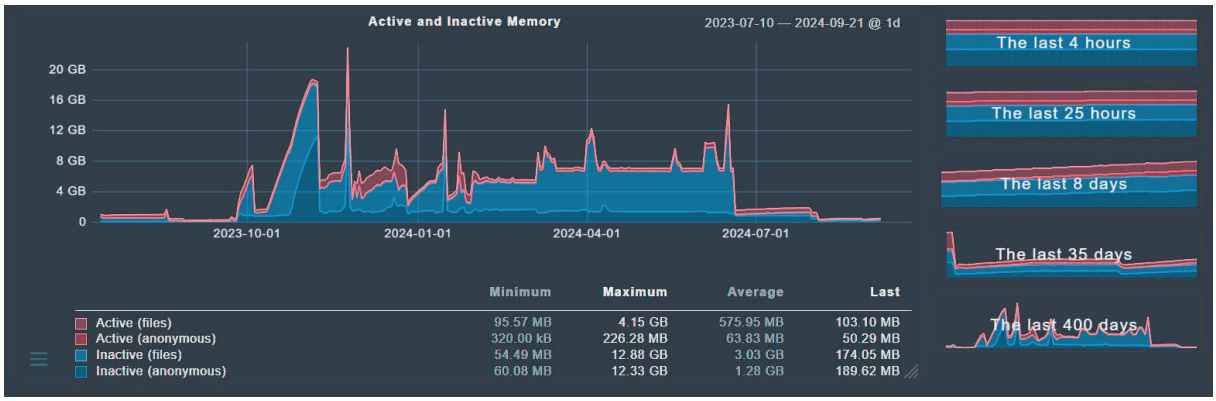


Figure 13 -DELIS Active vs Inactive Memory

The image shows a series of graphs representing active and inactive memory usage over different periods of time.

- Usage Trend: It is observed that the active memory (files) has a significant variation, with peaks reaching up to 4.15 GB. Idle (file) memory shows a maximum usage of 12.88 GB, suggesting that a significant portion of the memory is being dedicated to temporarily unneeded data.
- Usage Distribution: Active and inactive memory is used differently, with spikes that may indicate periods of high system usage.
- Resource Management: Continuous monitoring of memory usage helps identify and optimize areas of memory management, ensuring the system runs efficiently.

Effective memory management is crucial to prevent slowdowns and bottlenecks. The data suggests that the system has dynamic memory management, with significant variations in the use of active and inactive files. Continuing to monitor this data will help keep the system performing and prevent possible inefficiencies.

Funded by Horizon Europe Framework Programme of the European Union

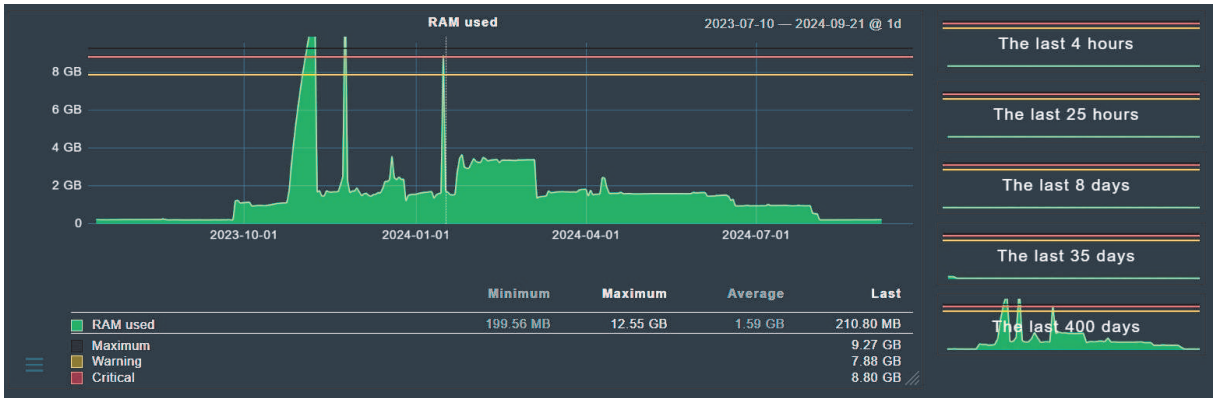


Figure 14 - DELIS RAM Used

The image shows a series of graphs and statistics relating to the use of the. The main graph on the left displays RAM usage with the y-axis representing the amount of RAM used in gigabytes (GB) and the x-axis representing dates. The green line indicates RAM usage, with significant peaks around October 2023 and January 2024. The highest recorded RAM usage is 12.55 GB, the lowest is 199.56 MB, and the average is 1.59 GB. The last recorded RAM usage is 210.80 MB.

- Usage Trends: Significant peaks are observed during the months of October 2023 and January 2024, suggesting periods of increased activity or intensive workloads.
- Usage Distribution: RAM used varies considerably, with a maximum of 12.55 GB. This may indicate times of high system usage.
- Resource Management: The main and additional graphs provide a clear view of how RAM has been used over time, allowing you to identify trends and potential problems related to memory consumption.

Monitoring RAM usage is crucial to ensuring optimal system performance and preventing bottlenecks. The data suggests that, while there are periods of high use, overall resource use appears to be under control. Continuing to monitor this data will help keep the system efficient and quickly identify any emerging issues.

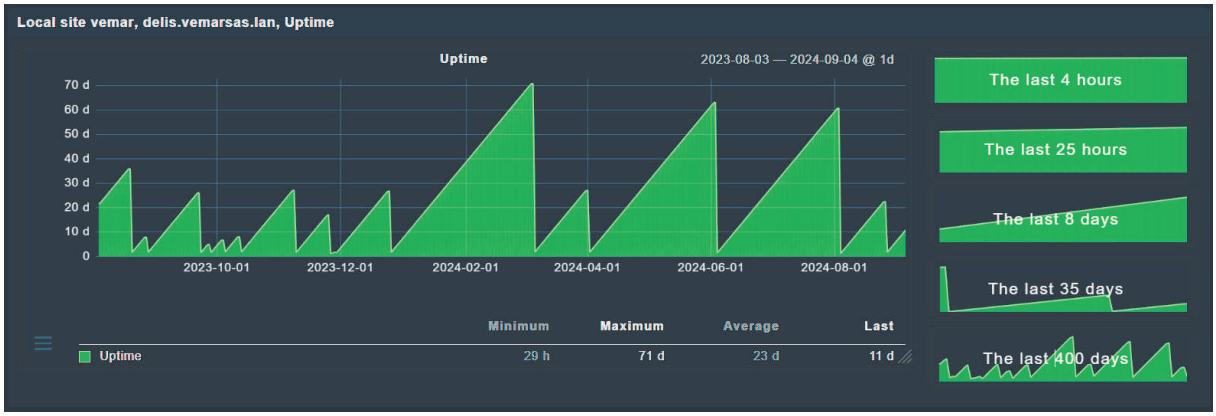


Figure 15 - DELIS UpTime

The image is a graphical representation of the operational uptime before an outage. The graph shows uptime in days on the y-axis and date on the x-axis. Uptime is

Funded by Horizon Europe Framework Programme of the European Union

represented by green triangular peaks, with the highest peak reaching 71 days. The minimum recorded uptime is 29 hours, the maximum is 71 days, and the average uptime is 23 days. The last uptime shown is 11 days.

- Uptime Trends: Activity peaks highlight prolonged periods of continuous operation, with the maximum peak being 71 days, indicating system stability for long periods without reboots.
- Stability and Reliability: The average uptime of 23 days suggests that the system is generally stable, although there have occasionally been reboots that have brought the average down.
- Future Considerations: Continuously monitoring and analyzing uptime helps identify any recurring issues that cause reboots and further improve system stability.

Maintaining high uptime is indicative of a stable and well-managed system. The data suggests that, although there were occasional reboots, the system generally maintained a good level of continuous operation. Continuing to monitor this data can help you identify areas for improvement and maintain high system reliability.

KPI 2.3

Microservices metrics will provide insights into the behavior and performance of microservices in a Kubernetes pod

FLUIDOMOS MEASUREMENTS
SYSTEM = FLUIDOMOS
DATE = 20/01/2025
host = Mixed: Vemar Local Site + Remote Provider
checkmk name = fluidomosv2.vemarsas.it

Among the various study and test systems, the one used for the present analysis is composed of two clusters, cunsumer and provider (consumer1-fluidomos and provider1-fluidomos), whose information data can be accessed through the following dashboard panels on Rancher (More details on Rancher provided in the Visualization Tool paragraph below).

State	Name	Provider	Kubernetes Version	CPU	Memory	Pods
Active	consumer	Imported K3s	v1.24.17+k3s1	17 cores	29 GiB	39/270
Active	consumer1-fluidomos	Imported K3s	v1.24.17+k3s1	34 cores	84 GiB	53/650
Active	local	Local K3s	v1.28.6+k3s2	8 cores	7.75 GiB	6/110
Active	provider	Imported K3s	v1.24.17+k3s1	16 cores	28 GiB	25/220
Active	provider1-fluidomos	Imported K3s	v1.24.17+k3s1	16 cores	58 GiB	22/220

Figure 16 - FLUIDOMOS Clusters

Funded by Horizon Europe Framework Programme of the European Union

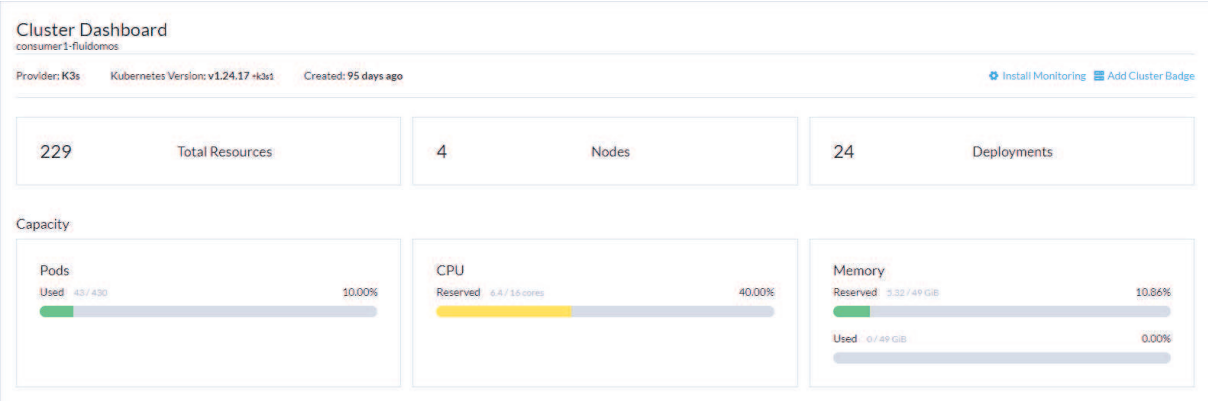


Figure 17 - FLUIDOMOS Consumer Dash Board

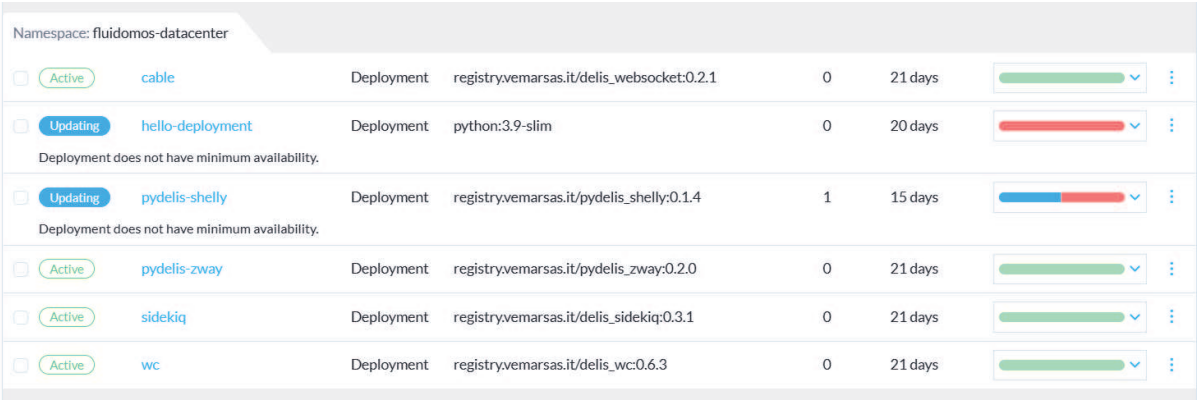


Figure 18 - FLUIDOMOS Namespaces

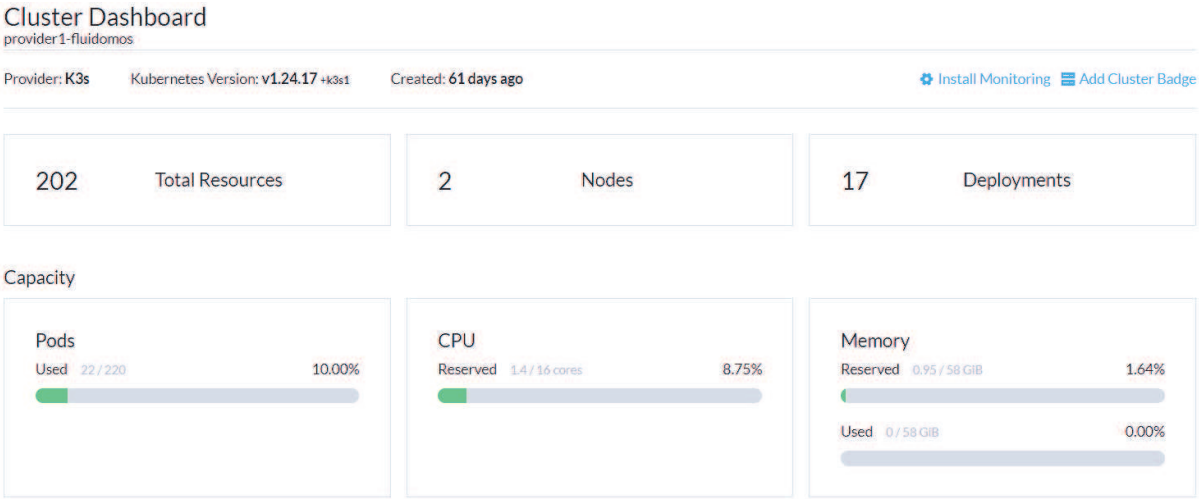


Figure 19 - FLUIDOMOS Provider Dash Board

Particular attention has been paid to consumer and service behavior. The following figures will analyze what has been derived from the measures.

Note A technical problem with the monitoring server caused a loss of information in a short period of measurement, visible in the graphs with the presence of a hole.

Funded by Horizon Europe Framework Programme of the European Union

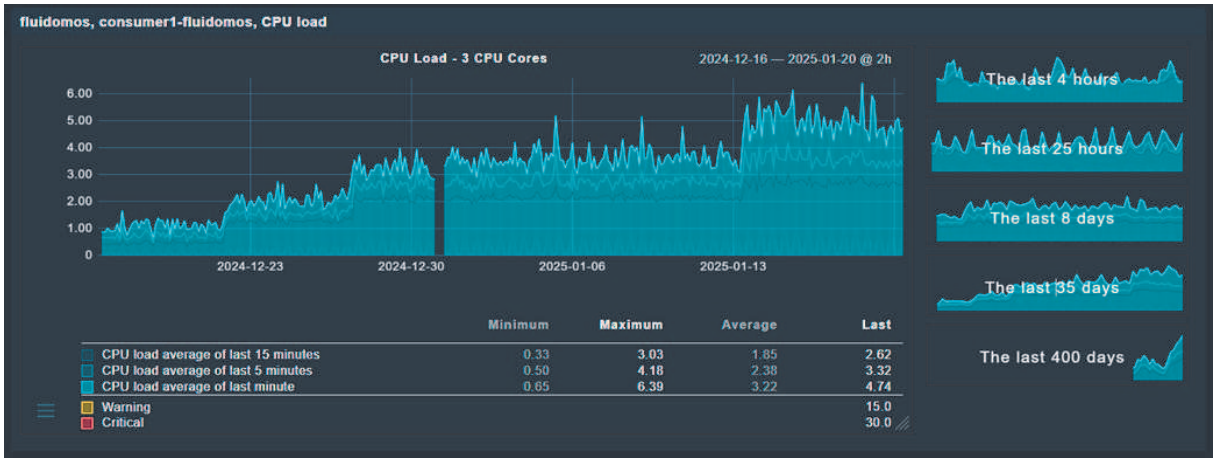


Figure 20 - FLUIDOMOS CPU Load

The chart tracks CPU load averages over time, highlighting values over the period from 2024-12-16 to 2025-01-20.
Load Averages:

- 15 minutes: Minimum: 0.33, Maximum: 3.03, Average: 1.85, Last: 2.62
- 5 minutes: Minimum: 0.50, Maximum: 4.18, Average: 2.38, Last: 3.32
- 1 minute: Minimum: 0.65, Maximum: 6.39, Average: 3.22, Last: 4.74

Between 2024-12-23 and 2024-12-30, the CPU load remains relatively stable, with minor fluctuations around 1.5–2.0. After 2025-01-01, there is a noticeable upward trend in CPU load, with peaks exceeding 4.0 in the 5-minute and 1-minute averages. The workload appears to intensify around 2025-01-13, where consistent peaks are observed in the graph.

About CPU Usage and Performance the system appears to operate well within the defined thresholds (Warning: 15, Critical: 30), indicating no signs of overload or performance issues at a high level. The increasing trend in CPU load after the new year shows the realignment of the resources to Delis System.

The System Behavior shows a consistent alignment between the 15-minute, 5-minute, and 1-minute averages sign that the load increases are sustained rather than temporary spikes. This might indicate prolonged computational tasks or resource-intensive operations.

© 2022-2025 FLUIDOS

Funded by Horizon Europe Framework Programme of the European Union

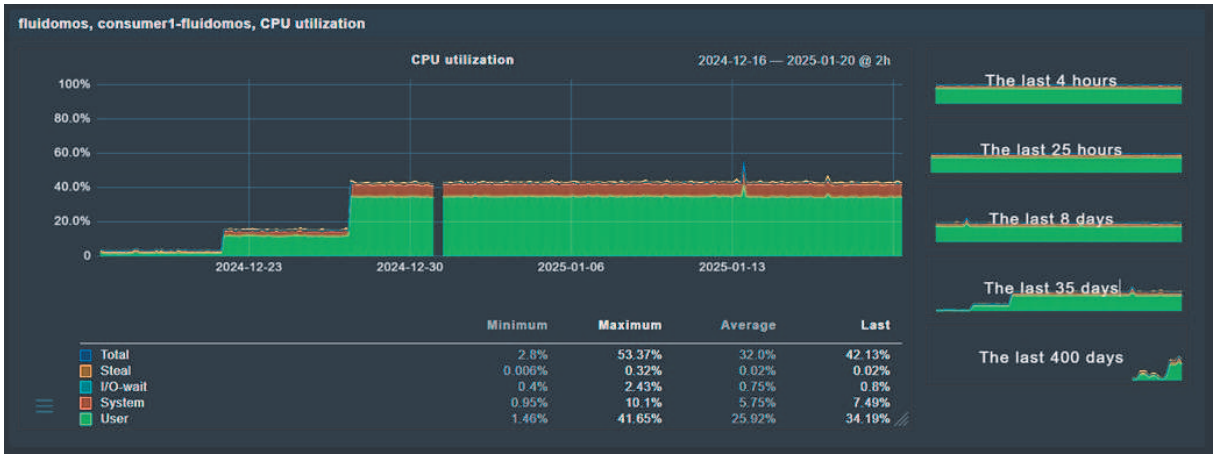


Figure 21 - FLUIDOMOS CPU Utilization

This graph represents the CPU utilization between 2024-12-16 and 2025-01-20, broken down into usage categories: User, System, I/O-wait, and Steal.
Total Utilization:

- Minimum: 2.8%, Maximum: 53.37%, Average: 32.0%, Last: 42.13%.

Detailed Breakdown:

- Steal: Negligible (<1%).
- I/O-wait: Min: 0.4%, Max: 2.43%, Avg: 0.75%.
- System: Min: 0.95, Max: 10.1%, Avg: 5.75%.
- User: Min:1,46, Max: 41.65%, Avg: 25.92%.

Before 2024-12-30, the total CPU utilization remains low, hovering around 10-15%. After 2024-12-30, there is a noticeable and sustained increase, with the total CPU utilization stabilizing at 40-50% on average. The graph shows very low I/O-wait and no steal time, indicating minimal contention for resources outside of CPU processing. The increase in utilization after 2024-12-30 aligns with a stated reduction in system resources, likely due to fewer CPU cores and reduced computational capacity being allocated to the system.

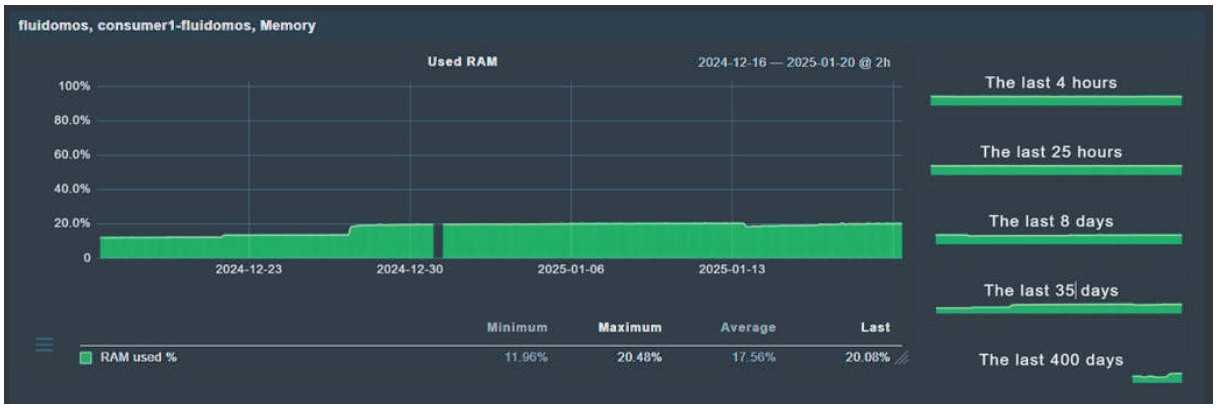


Figure 22 - FLUIDOMOS Used RAM

Funded by Horizon Europe Framework Programme of the European Union

The image presents a graph depicting RAM usage over time. It is a monitoring dashboard screenshot from the system performance monitoring tool. The graph shows the percentage of RAM used, ranging from 0% to 100%, across a timeline spanning several weeks.
The Time Range is : 2024-12-16 to 2025-01-20. This means the data spans from

X-axis (Time): The horizontal axis represents time, with dates marked at intervals.

Y-axis (RAM Usage): The vertical axis represents the percentage of RAM used, ranging from 0% to 100%.

Data Line: A green line plots the RAM usage percentage over time. The line fluctuates slightly, indicating variations in RAM usage throughout the period.

Summary Statistics:
- Minimum: 11.96%
- Maximum: 20.48%
- Average: 17.56%
- Last: 20.08%

Before 2024-12-30, memory usage remains consistently below 15%. After 2024-12-30, there is a clear stepwise increase in memory utilization, stabilizing between 20-21%. The memory usage plateaus after the increase, indicating that the system's workload or configuration remained stable during this period. The stepwise increase in memory usage corresponds with the stated reduction in system resources. This likely results from fewer available resources, causing the system to operate with a higher baseline memory usage relative to its capacity.

Despite the increase, 20.08% RAM utilization is still relatively low, leaving sufficient headroom for additional processes or workload spikes. The system demonstrates efficient memory management, as no significant fluctuations or memory leaks are observed over time.

In summary, This figure demonstrates that memory usage, despite the reduction in allocated resources, remains well below critical thresholds. The stable plateau at approximately 20% utilization indicates efficient system behavior and sufficient headroom for accommodating additional workloads.

Funded by Horizon Europe Framework Programme of the European Union

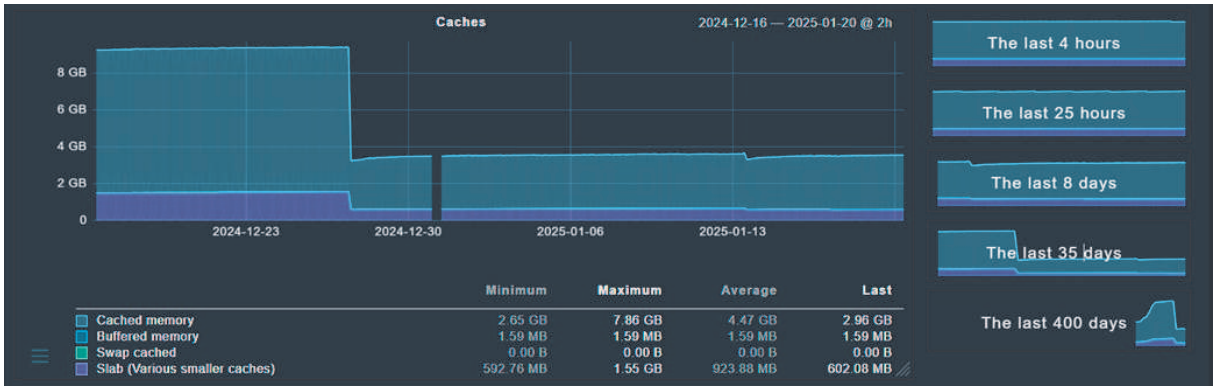


Figure 23 - FLUIDOMOS Caches

The graph presents a relatively stable pattern of cache memory usage over the observed period. There are no significant spikes or drops in the data, indicating a consistent level of cache utilization. The flat nature of the graph suggests a fairly constant workload on the system. This means that memory demands have been predictable, allowing the cache to maintain a steady level of utilization. The cache configuration, including size and replacement algorithms, appears to be well-suited to the workload. This has enabled the cache to operate efficiently, minimizing cache misses and maintaining a high hit rate. There is also no evidence of exceptional events, such as sudden spikes in load or hardware failures, that could have caused significant fluctuations in cache usage. Based on the available information, the system has exhibited very stable cache memory usage over the observed period indicating good performance and an optimal configuration.

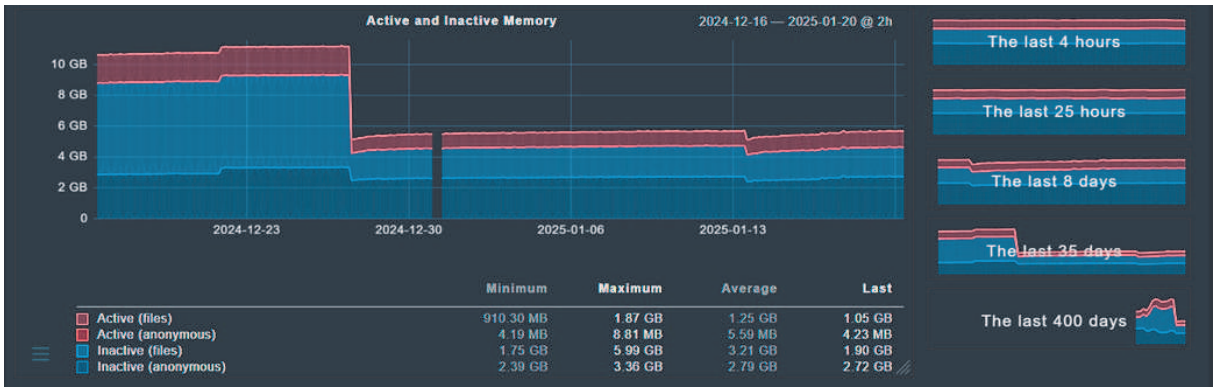


Figure 24 - FLUIDOMOS Active vs Inactive Memory

The graph provides a visual representation of active and inactive memory usage over a specific time period. The memory is categorized into two types: "files" and "anonymous".

Active Memory: This represents data that is currently being used by the operating system and applications. Fluctuations in active memory reflect changes in the system's workload.

Funded by Horizon Europe Framework Programme of the European Union

Inactive Memory: This includes data that is not currently in use but may be needed in the future. It's like a long-term cache.

"Files" and "Anonymous" Categories:

- Files: This refers to memory used to map files from disk into memory. Fluctuations in this area might indicate intensive file read/write operations.
- Anonymous: This category encompasses dynamically allocated memory, such as application heaps. Fluctuations here could indicate variations in memory usage by applications.

The pronounced drops in active memory, particularly in the "files" category, coincide with the reduction in system resources. This strongly suggests that the system is actively offloading data from memory to disk (paging) when resources are constrained, to make room for more critical processes. While the overall trend of inactive memory remains relatively stable, there might be slight increases around the resource reduction points. This could be due to pages being swapped out to disk and remaining in an inactive state until they are needed again.

The system likely adjusts its workload in response to resource constraints. For example, it might prioritize critical processes, leading to the termination or suspension of less important tasks. This can result in a decrease in active memory. The cache might be more aggressively evicting data during periods of resource scarcity. This is a common strategy to free up memory for more immediate needs.

The graph clearly shows the impact of resource constraints on memory usage. By understanding these relationships, system administrators can take steps to optimize system performance and prevent issues related to memory scarcity.
In summary, the image shows a system with well-balanced and stable memory usage, with a clear distinction between active and inactive memory. There are no indications of memory problems or system stress.

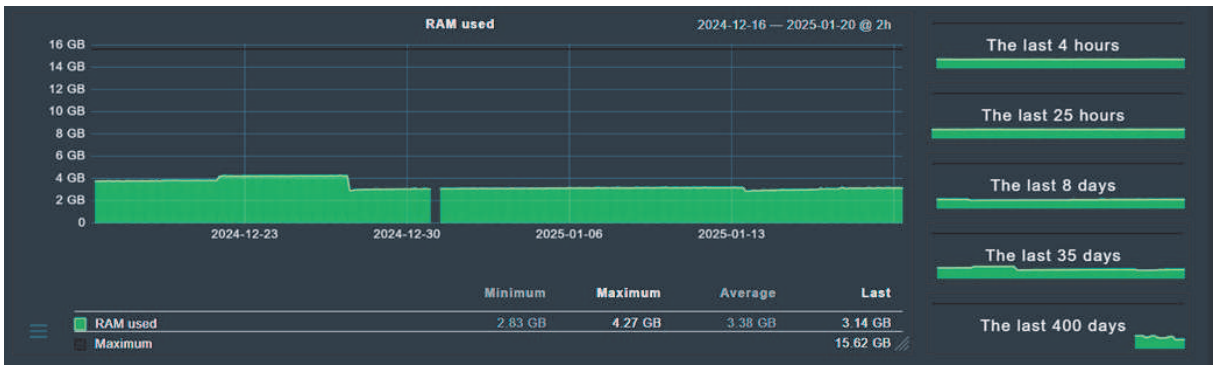


Figure 25 - FLUIDOMOS RAM Used

The graph presents a dynamic pattern of RAM usage, with notable fluctuations, especially around the points where system resources (CPU and RAM) were reduced.

Funded by Horizon Europe Framework Programme of the European Union

The pronounced drops in RAM usage coincide with the reduction in system resources. This strongly suggests that the system is actively offloading data from memory when resources are constrained, to make room for more critical processes. The system likely adjusts its workload in response to resource constraints. For example, it might prioritize critical processes, leading to the termination or suspension of less important tasks. This can result in a decrease in RAM usage. The cache might be more aggressively evicting data during periods of resource scarcity. This is a common strategy to free up memory for more immediate needs.
The graph clearly shows the impact of resource constraints on memory usage. The lack of sudden fluctuations in memory usage is indicative of several positive aspects of memory management. A consistent memory usage pattern reflects a relatively stable workload. This suggests that the applications running on the system have predictable memory requirements. The system's memory allocator is likely doing a good job of allocating and deallocating memory as needed. This implies that there are no significant memory leaks or excessive fragmentation. The cache is effectively holding frequently used data, reducing the need for frequent memory accesses. A well-tuned cache can significantly improve performance. Overall, a stable memory usage pattern indicates a robust memory management system that is effectively handling memory allocation and deallocation.

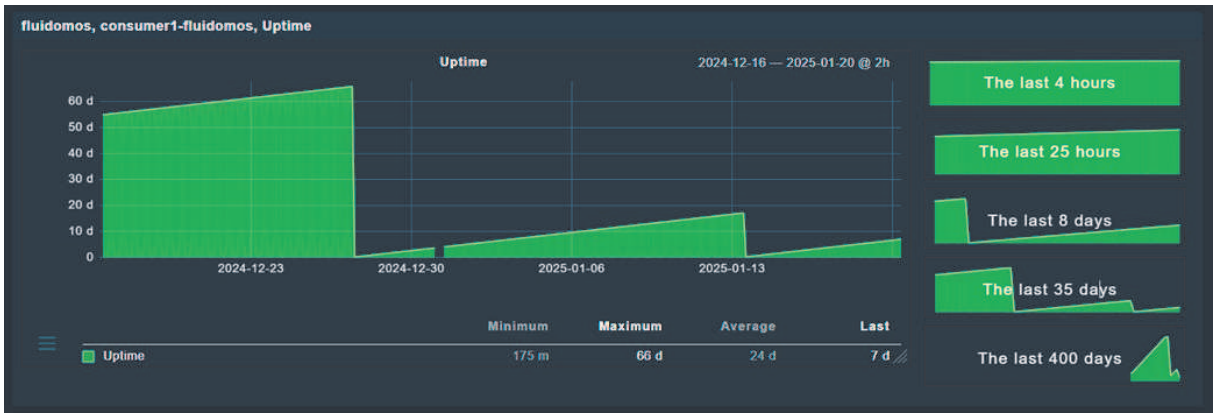


Figure 26 - FLUIDOMOS UpTime

The graph presents a visual representation of the system's uptime over a specific time period and shows up to 60 days of continuous activity. It shows a series of plateaus and drops, indicating periods of continuous operation followed by restarts or system failures.The extended plateaus in the graph represent periods of uninterrupted system operation. The length of these plateaus indicates the system's reliability during those intervals. The abrupt drops in the graph signify system restarts or failures. The frequency of these drops suggests potential issues with system stability or reliability. Drops are due to maintenance activities and related restarts due to rearrangement of the system resources and changes to system configuration.

Scheduled nature of the downtime reflects planned maintenance and a failure in the server powering. This shows a reduced risk of system failures. The system exhibits a high degree of reliability between drop windows, as evidenced by the extended uptime periods. The regular scheduling of maintenance tasks then, can

Funded by Horizon Europe Framework Programme of the European Union

demonstrate a proactive approach to system management, aiming to prevent issues rather than simply reacting to them. The duration of maintenance windows might vary depending on the complexity of the tasks being performed. The frequency of maintenance activities is likely determined by factors such as system criticality and the rate of change in the system's environment. While planned maintenance is necessary, it can temporarily disrupt services therefore, it's important to schedule maintenance during off-peak hours or to minimize the impact on users.

The graph indicates a well-managed system with a strong focus on proactive maintenance. The regular and planned nature of the downtime suggests a mature IT operations team that is committed to ensuring system reliability and availability.

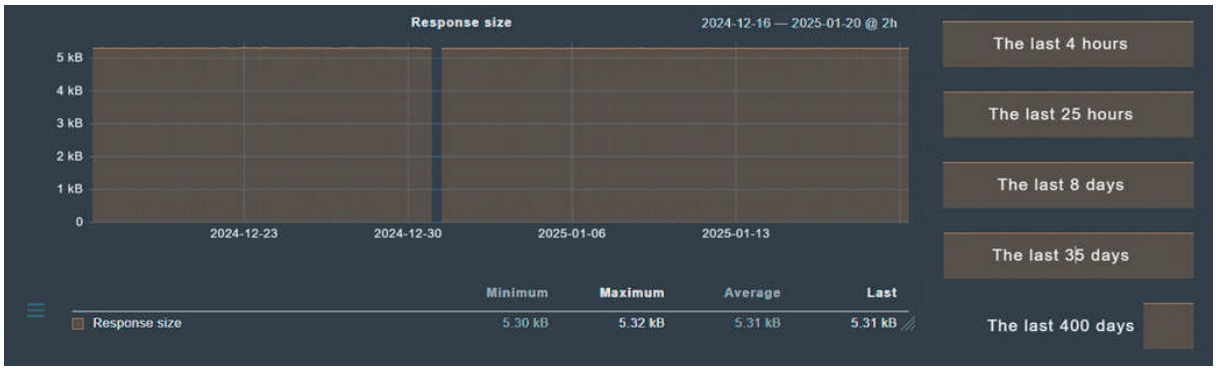


Figure 27 - FLUIDOMOS Response Size

The graph depicts the size of responses over a specific time period, exhibiting a relatively stable pattern with minimal fluctuations. The consistent nature of the response sizes suggests a stable system and process generating these responses. This implies that the underlying factors influencing the response size remain relatively unchanged throughout the monitored period. The small fluctuations around the average value indicate that there are likely no significant events or changes affecting the response size. This could suggest a well-tuned system or process with consistent performance.
The stable response size pattern suggests a high degree of system stability and reliability. With consistent response sizes, it becomes easier to predict system performance and plan for future capacity needs. A stable response size indicates that the system is efficiently utilizing its resources, avoiding excessive overhead or underutilization.
The graph indicates a well-behaved system with consistent and predictable response sizes. This suggests that the underlying processes are functioning efficiently and reliably.

Funded by Horizon Europe Framework Programme of the European Union

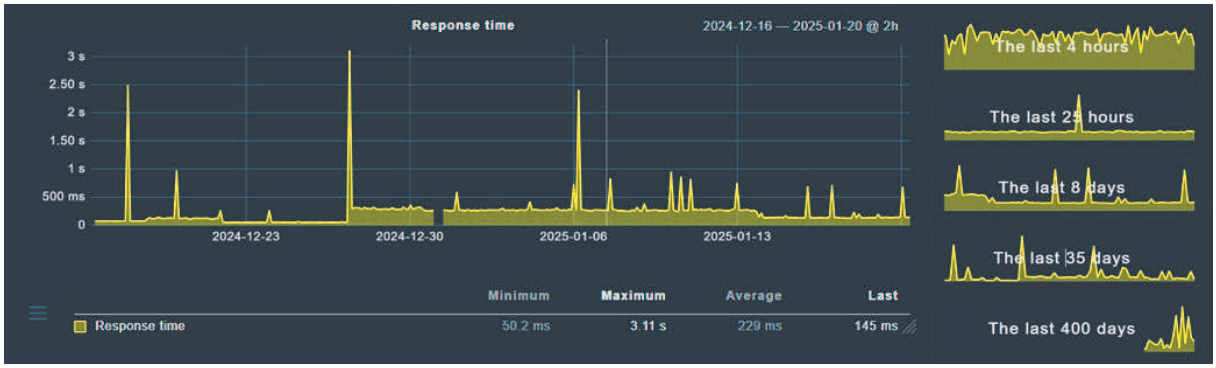


Figure 28 - FLUIDOMOS Response Time

The graph presents a visual representation of the system's response time over a specific period. It demonstrates significant fluctuations, with spikes exceeding 3 seconds interspersed with periods of lower latency. The pronounced peaks in response times suggest the presence of bottlenecks, network delays, or temporary system problems causing substantial slowdowns. Despite the spikes, there are also periods of relative stability with lower response times, indicating that the system can operate efficiently under certain conditions.
The response time graph clearly indicates the need for a more in-depth analysis to identify the root causes of the slowdowns and improve system performance. A thorough examination of data and the implementation of corrective measures can contribute to a smoother and more reliable user experience.

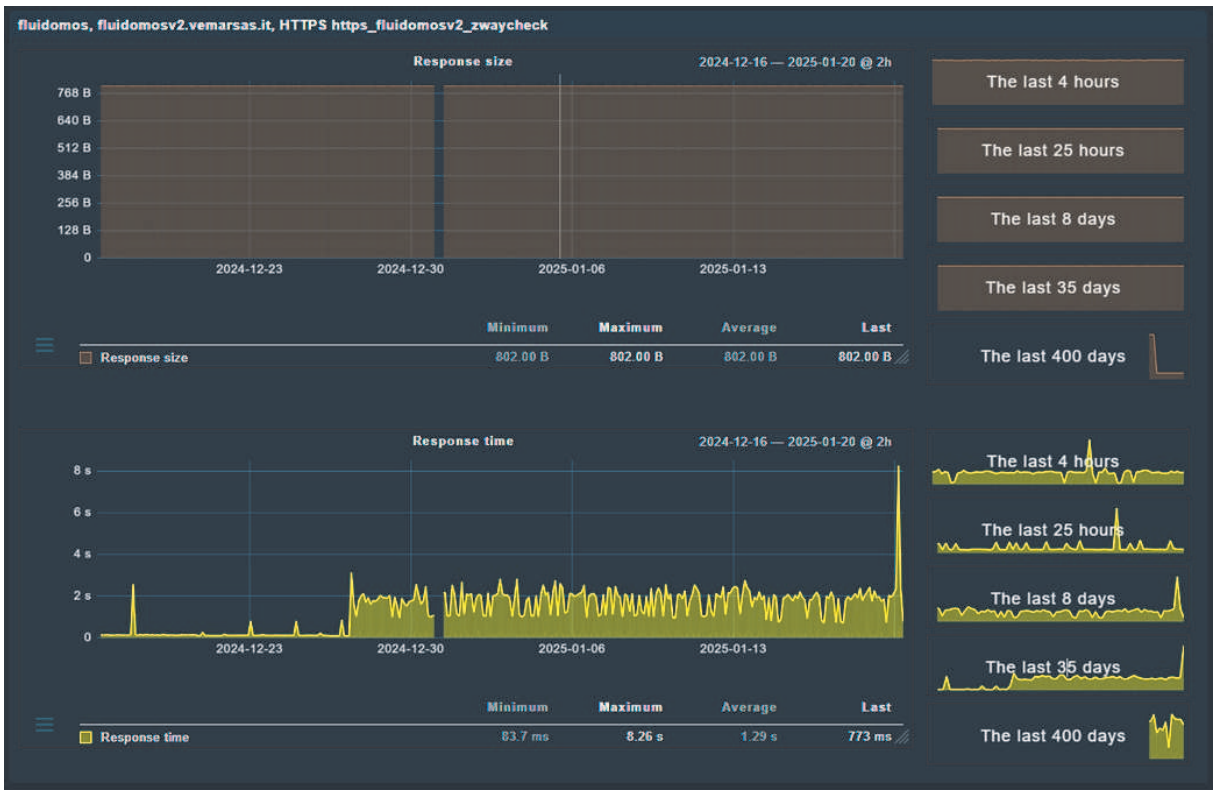


Figure 29 - Zway Service Check

Funded by Horizon Europe Framework Programme of the European Union

This graph shows the monitoring screen of one of the active services in the Fluidomos environment. The first one migrated from DELIS.

Response Size Graph: the response size remains almost perfectly constant throughout the monitored period. This high degree of consistency suggests the service and its data output have not undergone significant changes, the underlying processes generating the responses are very stable and the lack of variation implies a low occurrence of errors or exceptions that might alter the response size.
Consequently, the predictability of the response size can be advantageous for system resource planning and optimization, the consistent size indicates a robust system that is not susceptible to significant fluctuations due to varying workloads or external factors.

Response Time Graph: the response time exhibits substantial fluctuations, with frequent spikes. This indicates the service likely experiences varying levels of demand, causing fluctuations in response time, factors such as network conditions, system load, or resource contention can influence response times and the spikes suggest the presence of bottlenecks or inefficiencies in the system to be investigated.

In conclusion, while the service provides a consistent output size, the variability in response times suggests that further optimization is needed to ensure a reliable and performant user experience. By systematically investigating the factors contributing to the response time fluctuations, it is possible to identify and implement the necessary improvements.

Milestone M3

FLUIDOMOS tested and validated.
The pilot site validation will indicate that FLUIDOMOS has been implemented and tested in a real operational environment. Validation on this pilot site will be essential to compare the system's expected behavior with its actual execution in real conditions.
The results obtained during the validation phase will be shared, including the performance of the system and the resolution of any problems detected. The produced documentation will indicate how users can customize and configure their home automation systems using FLUIDOMOS. This may involve defining automation rules, configuring devices, integrating new devices, and more.

KPI 3.1

Funded by Horizon Europe Framework Programme of the European Union

Accessory services for the implementation of the pilot site using FLUIDOMOS are implemented

Within the FLUIDOMOS project, for the activities related to KPI 3.1, the implementation of new accessory services for additional functionalities was planned, in order to validate the integration mechanisms on the pilot site within the FLUIDOMOS environment using FLUIDOS.

A dedicated system with specific hardware and software was required.

The first 4 components listed in the following table were used from the acquired hardware

N	IOT COMPONENT	MANUFACTURER	DESCRIPTION
1	Revolution Pi Core S	KUNBUS	OS system
2	RevPi AIO	KUNBUS	Analog Input/Output (Voltage/Current)
3	RevPi DIO	KUNBUS	Digital Input/Output (Voltage/Current)
4	RevPi RO	KUNBUS	Relay Output expansion module
5	RevPi CAN	KUNBUS	CAN Bus Expansion Module
6	Berry IMU V3	Ozz Maker	Accelerometer, gyroscope, magnetometer, barometric/altitude sensor
7	Berry GPS-IMU V4	Ozz Maker	GPS, Compass, Temperature, Altitude

Table 9 : IOT Components

Remaining hardware will be considered for integration in the near future.

© 2022-2025 FLUIDOS

Funded by Horizon Europe Framework Programme of the European Union

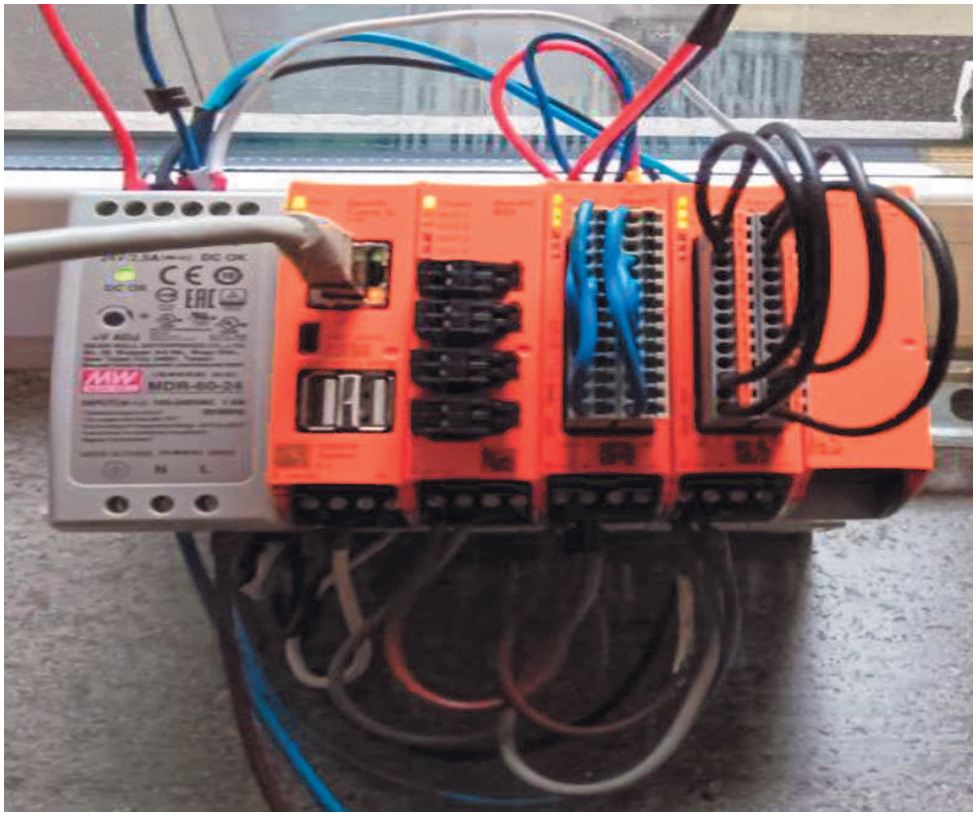


Figure 30 - IOT Expansion Modules and Core

In addition to a dedicated power supply, the hardware was installed on a DIN rail, and the following tasks were carried out:

- Wiring for power supply connections.
- Additional wiring for simulating sensors on the DIO module.
- Additional wiring for simulating sensors on the AIO module.

Test Configuration

From the configuration perspective, the following steps were performed:

- Booting the system in configuration mode.
- Installing the operating system.
- Updating the operating system.
- Configuring the expansion modules to ensure proper hardware recognition.
- Updating the drivers for interaction with the expansion modules.
- Verifying interaction modes with the expansion modules.
- Implementing a web interface for module interaction and configuring the corresponding ports for communication routing.
- Testing interaction with the modules.

Implementation of the Specific Plugin

Funded by Horizon Europe Framework Programme of the European Union

Every web service in DELIS is structured into two main parts: a generic framework and a specific plugin that implements the service logic. In this case, we are working on the plugin to acquire data from Revolution Pi (RevPi) modules. To access the module data (analog inputs, digital inputs, and electronic outputs), the plugin will use the PiControl driver, which is natively supported by the RevPi Core S and allows direct data read and write calls to the kernel. This approach ensures a fast and efficient connection to the expansion modules, leveraging the RevPi hardware architecture.

Execution of the Web Service on RevPi Core S

The web service will run directly on the RevPi Core S, which will act as an edge node in the FLUIDOMOS architecture. This means that the RevPi will not only act as a data acquisition device but as a node capable of exposing RESTful services externally. Commonly used technologies include Python frameworks such as Flask or FastAPI to build the service and PiServer SDK for specific RevPi management.

Exposing the Web Service Externally

Once completed, the web service will be exposed via RESTful APIs or an HTTP/HTTPS endpoint accessible over the network. The API will be designed to send data collected from the RevPi expansion modules to the DELIS system. Standard API documentation will be implemented to facilitate integration and maintenance.

Integration

FLUIDOMOS will use the exposed web service to read and write data. The integration will be configured by registering the plugin as a resource of the distributed cluster, leveraging DELIS's native support for microservices. The plugin will need to Handle bidirectional calls (data reading and writing) and Synchronize changes with replicated PostgreSQL tables, as described in previous examples, to ensure resilience in case of disconnections.

Management within the FLUIDOMOS Architecture

The new web service will be registered as a microservice within the FLUIDOMOS environment. FLUIDOMOS will act as an orchestrator, enabling Monitoring of the service's performance and Management of snapshots and backups to ensure persistence and rollback capability and Additionally, FLUIDOS could use the service for communication between distributed nodes, making the system more scalable.

© 2022-2025 FLUIDOS

Funded by Horizon Europe Framework Programme of the European Union

Advantages of the Approach

Resilience: The distributed architecture ensures that even in the event of a temporary disconnection from DELIS, data will be stored locally on the RevPi.

Scalability: The modular design allows for the easy addition of new services or edge nodes with similar functionalities.

Efficiency: PiControl enables direct and efficient access to RevPi data, reducing latency.

Interoperability: The service can be extended to interact with systems other than DELIS, thanks to the use of standard APIs.

Future Steps

Web Service Testing: Local verification of the service exposure and correct data acquisition via PiControl.

Deployment on RevPi Core S: Service deployment and network configuration for access.

Integration with DELIS: Plugin configuration in DELIS and communication testing.

Monitoring with FLUIDOMOS: Adding the service to the FLUIDOMOS network to ensure monitoring and management.

KPI 3.2

The produced documentation reports the technical analysis of performances and scalability.

DELIS VS FLUIDOMOS

Initially, independent hardware configurations were adopted. While DELIS had the following characteristics

Funded by Horizon Europe Framework Programme of the European Union

Delis

SalernoSilos
16GB RAM
8CPU

Fluidomos, after a series of modifications, came to have the following ones

Fluidomos (shared Resources)

StrasburgCluster - Consumer			SlernoCluster Provider	
Control Plain	Worker	Edge	Control Plain	Worker
16GB RAM	8GB RAM	Negligible	24GB RAM	
3CPU	3CPU	Negligible	6CPU	

To have better basis for comparison, it was decided to remodulate delis resources to

Delis

SalernoSilos
32GB RAM
12CPU

To make the comparison between the delis and Fluidomos systems, only the consumer control plane of the main Fluidomos cluster was taken as a reference. In it, the resources are limited compared to those of delis (3CPU vs 8CPU), but the use of the consumer workers and the provider cluster make up for this deficit.

CPU Load

Let's compare the two graphs showing CPU Load on different software systems, one referring to delis and the other to Fluidomos

Funded by Horizon Europe Framework Programme of the European Union

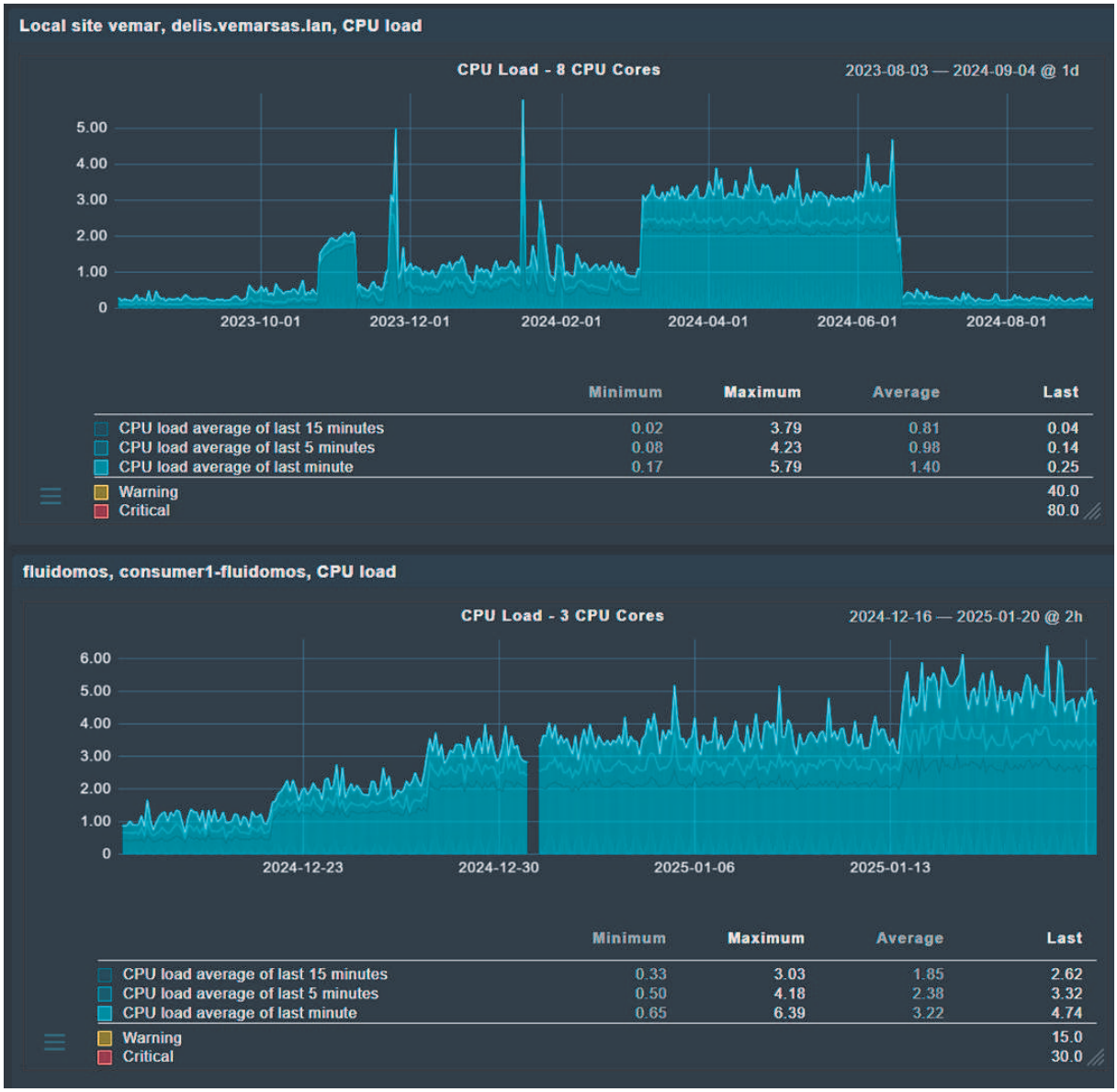


Figure 31 - CPU Load Comparison

Delis graph:

- Maximum recorded load: Peak value: 5.79 (on 8 cores); Very high spikes, but occasional and concentrated in specific intervals.
- Average CPU load: Overall average: 1.40; CPU activity fluctuates between periods of almost complete idleness and very intense loads.
- Minimum load: Minimum of 0.02, indicating the system is often almost idle.

Fluidomos graph

- Maximum recorded load: Peak value: 3.39, higher than delis but this occurs only after the reconfiguration of the resources; the load is distributed more evenly, with no extreme spikes.
- Average CPU load: Overall average: 3.22, higher than delis.
- Minimum load:Minimum of 0.33, indicating the system remains more active even during low usage periods.

Funded by Horizon Europe Framework Programme of the European Union

Delis shows more irregular and "spiky" behavior with wide oscillations and several periods of almost complete idleness (load near zero) followed by extremely high peaks. This may indicate a system responding to sudden loads or intensive batch jobs and/or load balancing issues or spikes linked to software/hardware inefficiencies and shorter periods of effective usage compared to Fluidomos.
Fluidomos instead has a more uniform and stable distribution of CPU load with no periods of complete idleness and behavior suitable for ensuring operational continuity and avoiding system stress peaks.
Fluidomos, apparently running on less capable or less optimized hardware for the software load result, resulting in lower peaks and idle periods. It utilizes cores in a more stable manner. Load optimization allows for more homogeneous distribution, avoiding both extreme peaks and prolonged idleness.

Fluidomos provides significant improvements compared to delis:

- Higher resilience: Uniform load distribution reduces the risk of interruptions due to overloads.
- Efficiency: Hardware appears to be used optimally, with no long idle periods.
- Peak reduction: Avoiding extreme peaks protects the hardware and improves overall performance.

Delis could improve through Load balancing (Avoiding extreme peaks and idleness periods) and Hardware/software upgrades (To better handle workload distribution).
Fluidomos results more stable, performant, and suitable for systems requiring continuous operation.

CPU Utilization

Funded by Horizon Europe Framework Programme of the European Union

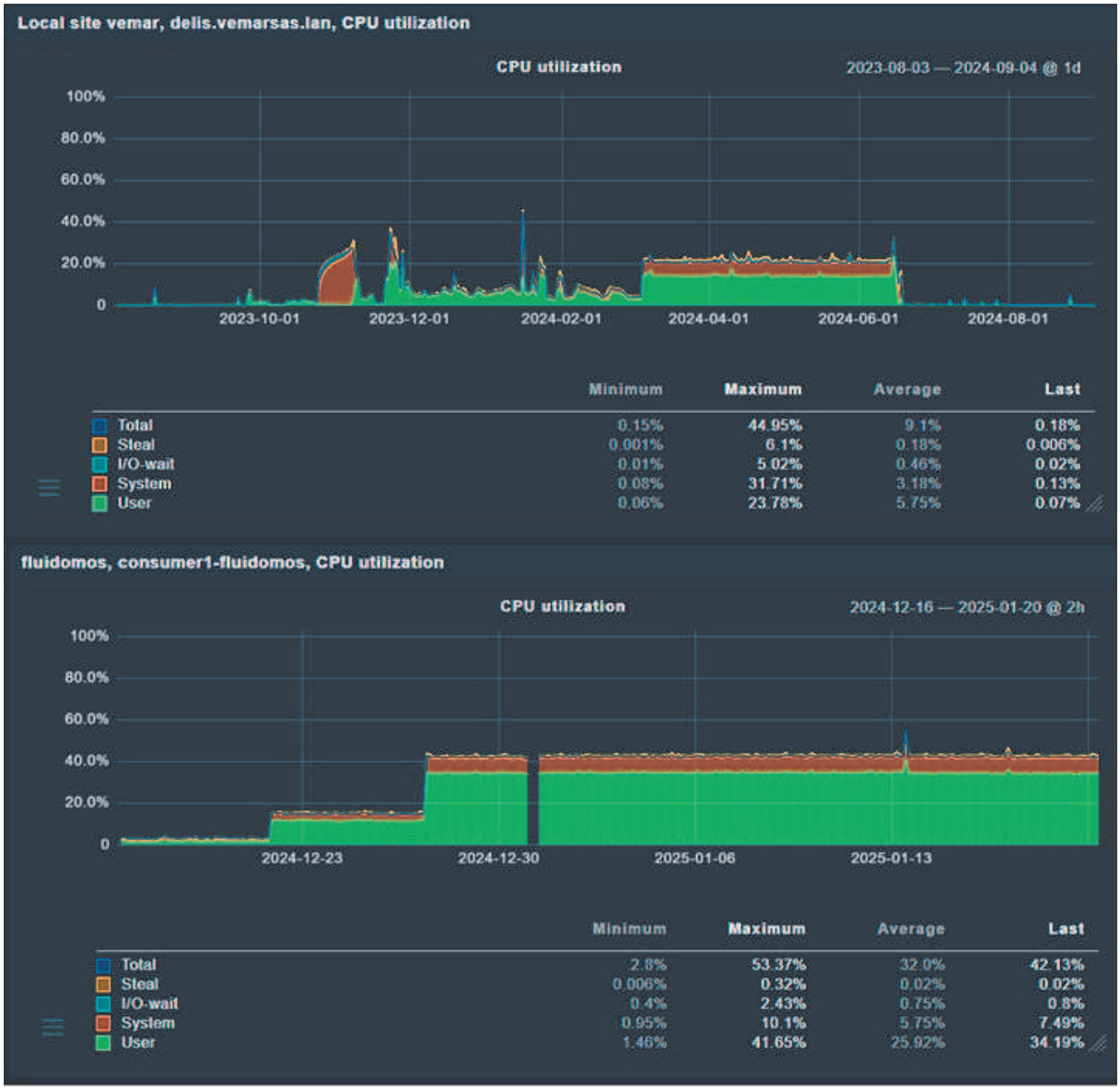


Figure 32 - CPU utilization Comparison

In Delis graph, the CPU peaks appear sporadically but are concentrated in specific intervals and the Maximum CPU utilization has a peak usage of 44.95%, a very high value, indicating periods of significant load. Overall average CPU utilization is 9.1%, relatively low compared to the maximum, suggesting most of the time the CPU is underutilized. The Graph indicates a User utilization with Peaks at 23.78% and average of 5.75% while a System utilization Peaks at 31.71% and average of 3.18%, showing the software frequently stresses the system. I/O wait results Relatively low at an average of 0.48%, though peaks at 5.02%, which could indicate moments of hardware bottleneck. Steal time is negligible (close to zero). Last value of 0.18%, indicating low activity at the end of the monitored period.

In the Fluidomos graph, the CPU utilization has Peak usage of 53.37% and average CPU utilization is 32.0%, higher than delis. High average utilization (32%) indicates better resource usage and consistent workload processing compared to DELIS. These occur after resource reconfiguration but demonstrate consistent resource

Funded by Horizon Europe Framework Programme of the European Union

usage. User utilization has Peaks at 41.67%, average of 25.92%,suggesting FLUIDOMOS is effectively leveraging its processing power for actual workloads.

Delis shows High variability, with some sharp peaks in CPU usage, but also long periods of very low activity (near 0%). Peaks suggest moments of intensive processing, possibly due to batch jobs or inefficient handling of system tasks. Periods of inactivity indicate suboptimal resource utilization and wasted potential.

Fluidomos has a more stable and predictable trend with limited spikes. Continuous stable activity, reflecting a more balanced and sustained workload. No prolonged idle periods, implying better utilization of available hardware.

Fluidomos demonstrates clear advantages such as

- Efficiency: Lower peaks and sustained activity reduce wear on the hardware and optimize power consumption.
- System stress: More stable average and system utilization suggest that hardware is less stressed.
- Resilience: Continuous, moderate usage ensures reliability and smooth operation.

Delis could improve through:

- Workload balancing: Avoiding sharp peaks and ensuring more consistent resource usage.
- Hardware upgrades: May help reduce bottlenecks and improve average performance.
- Software optimization: To ensure efficient task distribution and reduce I/O wait times.

While DELIS conserves resources during idle periods, it shows inefficiency in workload distribution. On the other hand, FLUIDOMOS demonstrates high efficiency in handling workloads.

FInally, Fluidomos clearly outperforms delis in terms of resource management, stability, and efficiency. Delis, while capable of handling intense workloads, exhibits inefficiencies and hardware stress, making it less ideal for continuous or balanced operations.

Used RAM

Funded by Horizon Europe Framework Programme of the European Union

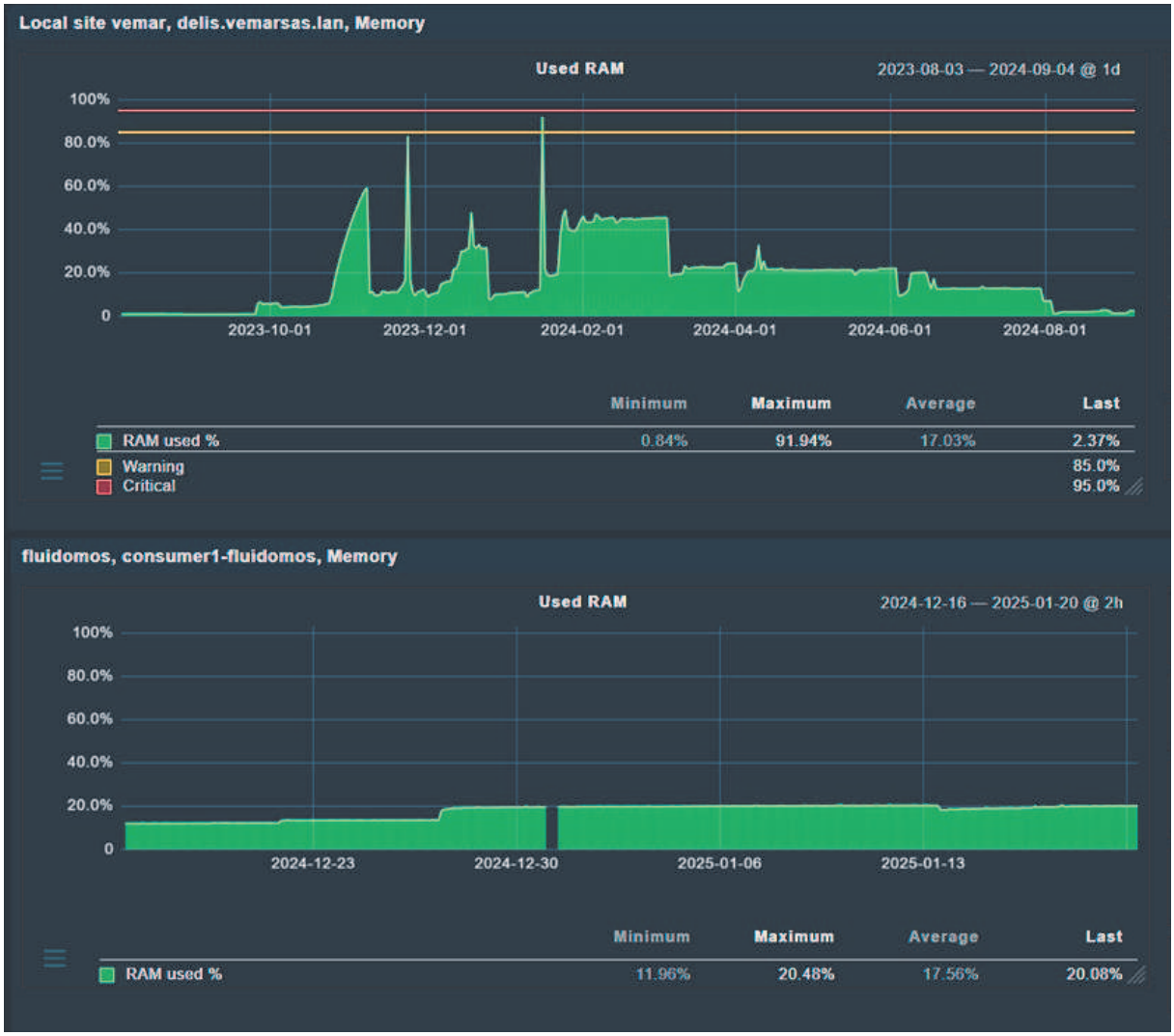


Figure 33 - Used RAM Comparison

In Delis graph, Maximum memory usage Peaks at 91.94%, very close to the warning threshold (85%), and the critical limit (95%). Such high peaks suggest moments of memory pressure, potentially due to resource-intensive tasks or poor memory optimization. Average memory usage is 17.03%, relatively low when considering the peaks, indicating most of the time the system is not heavily utilizing RAM. Minimum memory usage drops to 0.84%, showing substantial idle time or minimal workloads during certain periods. Last value is 2.37%, indicating a recent period of very low usage.

In Fluidomos graph, Maximum memory usage peaks is at 20.48%, significantly lower than delis, suggesting that memory is being efficiently managed. Average memory usage is 17.56%, a steady and moderate level, indicating a balanced workload. Minimum memory usage maintains a baseline of 11.96%, reflecting consistent resource usage even during low activity. Last value is 12.13%, showing stability with no drastic reductions or spikes.

Funded by Horizon Europe Framework Programme of the European Union

Delis memory usage fluctuates dramatically, with several sharp spikes reaching near-critical levels. Extended periods of very low usage suggest suboptimal utilization of available resources. The spikes could indicate memory leaks, inefficient software, or periodic execution of memory-heavy processes.
Fluidomos memory usage remains remarkably stable with no significant spikes also after system resource reconfiguration. The system consistently utilizes a portion of its RAM, which suggests efficient resource allocation and minimal overhead.

While Delis can improve by

- Software optimization: Investigate and optimize memory-intensive processes to reduce peaks.
- Monitoring memory leaks: Ensure there are no memory leaks causing usage to rise over time.
- Load distribution: Balance tasks to avoid large fluctuations and make better use of available RAM during idle periods.
- Hardware upgrade: If tasks are inherently memory-intensive, increasing RAM may help mitigate pressure.

Fluidomos seems to have no immediate improvements to be done as the system demonstrates efficient and balanced memory usage.

Fluidomos outperforms delis in terms of memory management, with lower, stable, and efficient usage across the monitoring periods while Delis suffers from irregular and extreme fluctuations in memory usage, suggesting potential inefficiencies in task scheduling or resource management.

© 2022-2025 FLUIDOS

Funded by Horizon Europe Framework Programme of the European Union

Caches

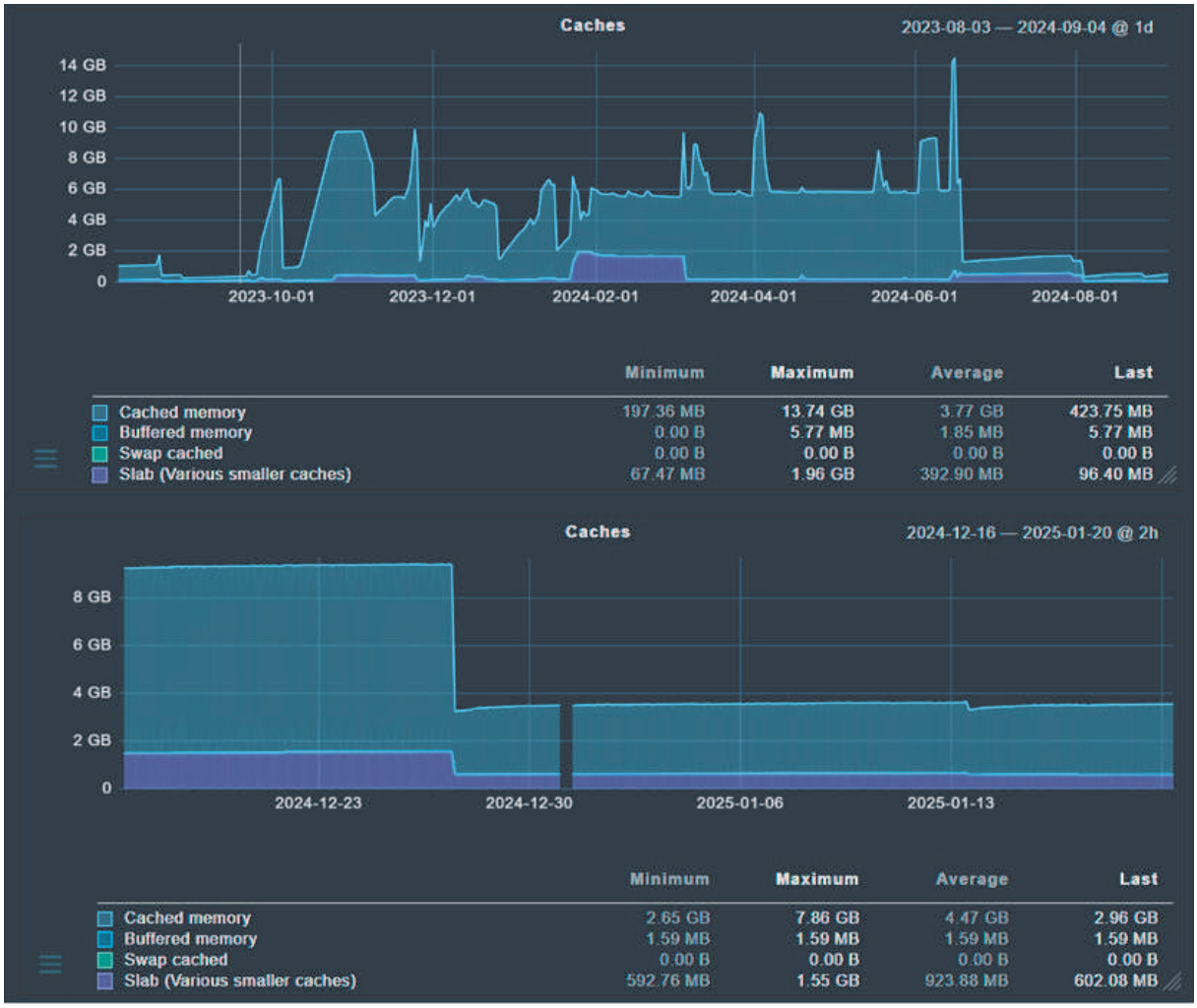


Figure 34 - Caches Comparison

Delis Cache Usage shows High variability, with significant spikes and drops in "Cached memory" usage. Peaks reach up to 14 GB, while troughs go down near zero. This indicates periods of intense caching activity followed by periods of low cache usage."Buffered memory" and "Swap cached" are consistently low (near zero), suggesting minimal use of these memory areas. "Slab" usage fluctuates but remains generally below 2 GB. The large fluctuations in "Cached memory" suggest a workload with varying demands on data access. This could be due to scheduled tasks, periodic data processing, or varying user activity. The low usage of "Buffered memory" and "Swap cached" indicates the system generally has enough RAM and doesn't rely heavily on disk caching or swapping.

Fluidomos Cache Usage indicates "Cached memory" is remarkably stable, hovering around 7.8 GB. There are almost no fluctuations. Other Metrics, similar to the first graph, "Buffered memory" and "Swap cached" are negligible. "Slab" usage is also very stable around 1 GB. The extremely stable "Cached memory" usage suggests a consistent and predictable workload. This could indicate a system performing a repetitive task or serving a constant stream of requests. The stable "Slab" usage further supports the idea of a stable and consistent system operation.

Funded by Horizon Europe Framework Programme of the European Union

In conclusion, the two graphs reveal distinct patterns of cache usage, indicative of different software and likely similar hardware configurations. First graph shows a system handling a dynamic workload, while the second graph depicts a system with a highly consistent and stable workload.

Active vs Inactive Memory

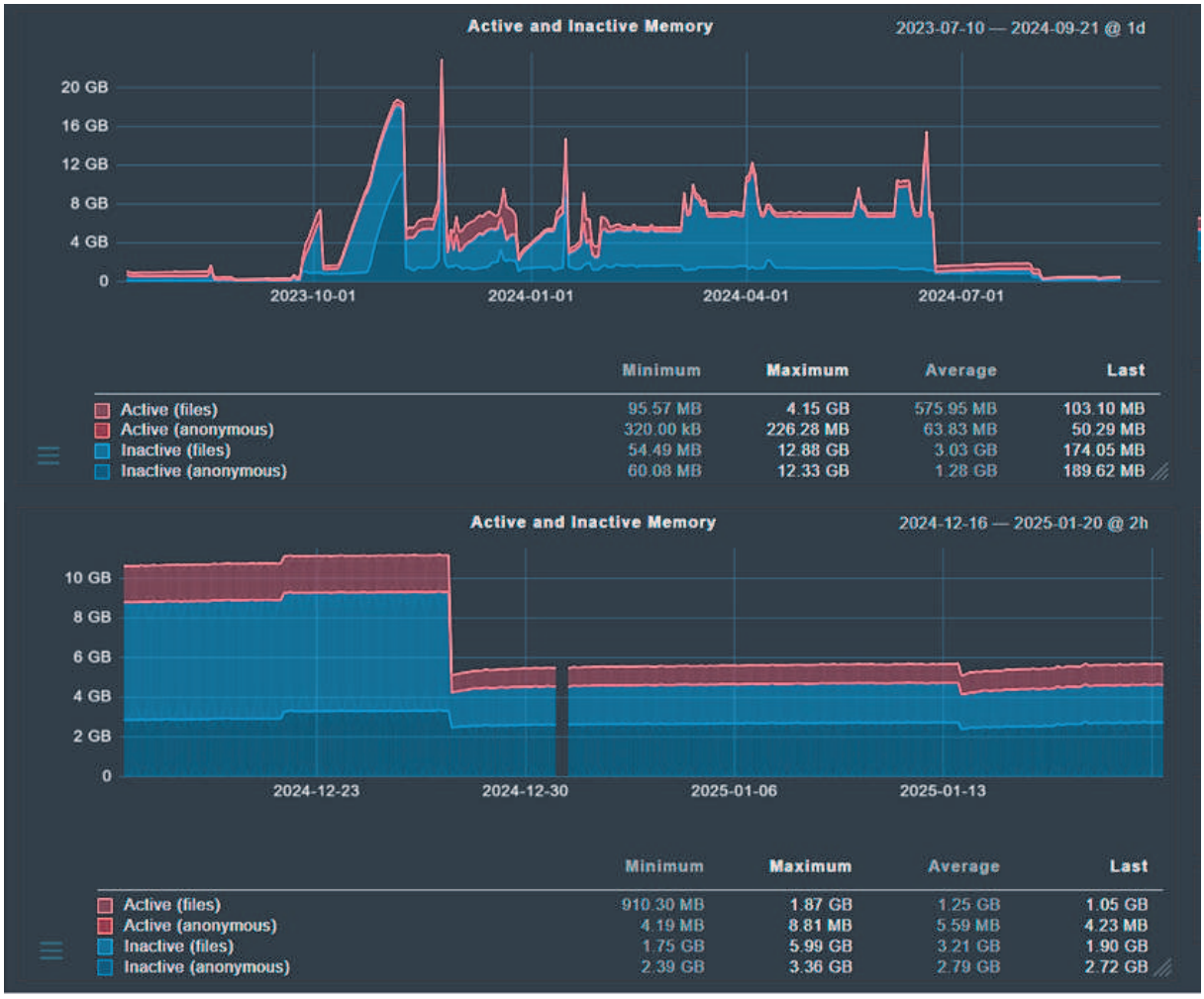


Figure 35 - Active & Inactive Mem Comparison

Delis Memory Usage shows considerable fluctuations in both active and inactive memory. Significant peaks are noticeable, especially in inactive memory (up to almost 13 GB), followed by periods of lower usage. Active memory shows more contained variations, but still present. Components can be divided in Active (files) that shows a relatively stable trend with some peaks; Active (anonymous) that remains at low values, almost invisible in the main graph; Inactive (files) with the most variable component, with wide oscillations; Inactive (anonymous) similar to "Inactive (files)," but with slightly less pronounced peaks. The large fluctuations in inactive memory suggest a variable workload, with periods of intense activity followed by periods of inactivity. This could be due to batch processes, backups, updates, or variations in system usage. The presence of peaks in both active and inactive memory indicates that the system manages files of different sizes and with variable access patterns.

Funded by Horizon Europe Framework Programme of the European Union

Fluidomos Memory Usage shows both active and inactive memory show an extremely stable trend. Variations are minimal. Active (files) shows only a decrease in resource reconfiguration around almost stable values. Active (anonymous): Very low and stable values; Inactive (files) and Inactive (anonymous) has the same behaviour of the previous around stable and lower values. The stability of all components indicates a very constant and predictable workload. This suggests a system performing a repetitive task or serving a constant stream of requests.

The two graphs show distinct memory usage patterns, indicative of different software and likely similar hardware configurations. Delis graph shows a system handling a dynamic workload, while Fluidomos graph shows a system with a very constant and stable workload.

© 2022-2025 FLUIDOS

Funded by Horizon Europe Framework Programme of the European Union

RAM Used

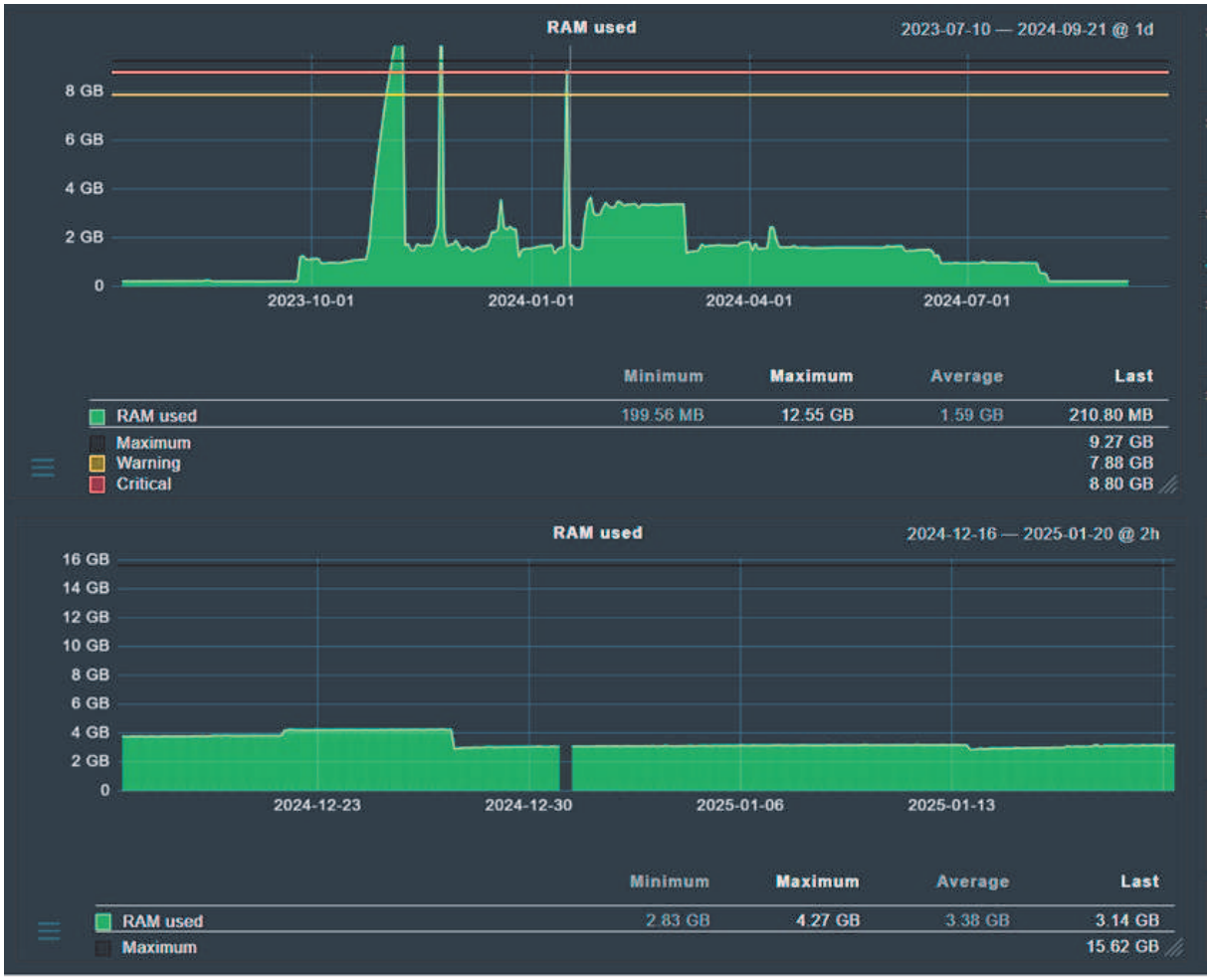


Figure 36 - RAM Used Comparison

DELIS RAM Usage Trend initially shows a relatively stable trend with low values. Subsequently, between October 2023 and January 2024, there are significant peaks reaching a maximum of 12.55 GB. After the peaks, a gradual reduction and stabilization of memory consumption are observed, which remains low until the end of the monitored period (last value: 210.80 MB). The peaks are likely linked to moments of particularly intense operational load, such as during the execution of heavy processes or updates. The final phase demonstrates a notable optimization in memory usage, suggesting possible software improvements or a reduction in the functionalities being executed.

FLUIDOMOS RAM Usage Trend exhibits a more stable behavior compared to DELIS. The used RAM ranges between 2.83 GB and 4.27 GB, with an average consumption of 3.38 GB. Visible "steps" correspond to deliberate reductions in the resources allocated to the system. The system's stability is indicative of a more controlled resource management. Even with resource reductions (the steps), the system appears to operate without evident issues. The last consumption value (3.14 GB) is close to the average, demonstrating operational consistency.

Funded by Horizon Europe Framework Programme of the European Union

Comparing DELIS and FLUIDOMOS, we can state that FLUIDOMOS shows more stable memory consumption which could indicate greater efficiency or optimized resource management. DELIS, on the other hand, displayed periods of intense load but later drastically reduced consumption, a sign of corrective actions.The steps visible in the FLUIDOMOS graph show that the system is designed to adapt to changes in resource availability while maintaining stable operation. DELIS, despite the initial peaks, demonstrates a significant reduction in consumption toward the end, indicating improvements in resource management. FLUIDOMOS demonstrates predictable and stable behavior, which might indicate a more robust design or a more mature operating environment.

In Conclusion, DELIS experienced an initial period of high consumption, which was later followed by a drastic reduction. However, during the peaks, it might have suffered from instability or inefficiency. FLUIDOMOS, despite the reductions in available resources, maintains operational stability, which is a positive indicator of the system's robustness.

© 2022-2025 FLUIDOS

Funded by Horizon Europe
Framework Programme of the European Union

UpTime

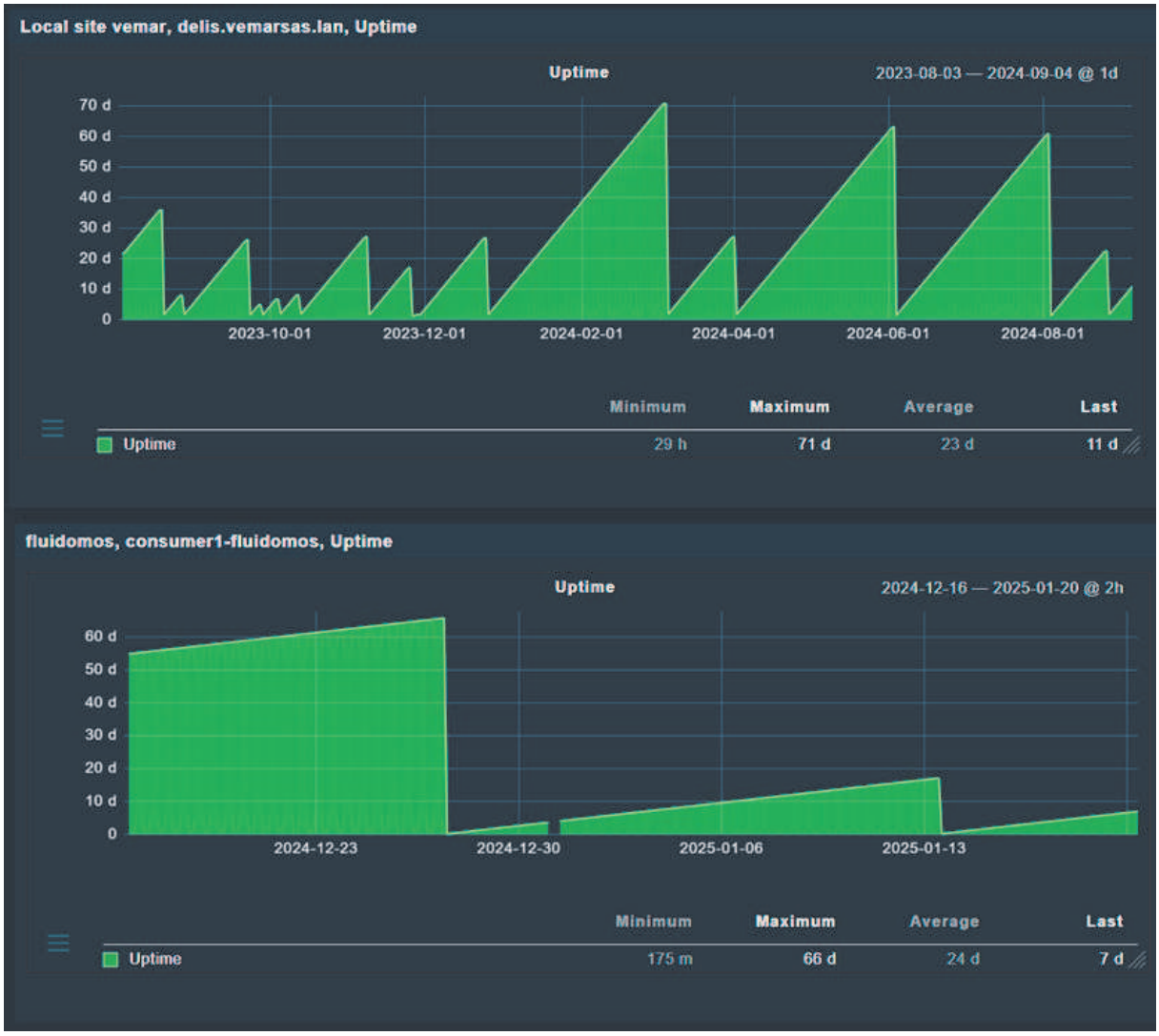


Figure 37 - Up Time Comparison

Looking at the two graphs illustrating the uptime chart of the two systems, DELIS shows frequent resets or downtime, with uptime capped at approximately 71 days; most of them are due to Maintenance, hardware limitations, or instability in the system that might cause interruptions;
FLUIDOMOS instead, in the analyzed period, has a consistent and similar growth in uptime, reaching 66 days without resets prior of the first restart for planned hardware reconfiguration; as a result there is an improved system stability, potentially due to better hardware/software configurations and fewer interruptions.
Fluidomos demonstrates improved resilience and reliability. Transitioning DELIS to this configuration enhances operational continuity.

Funded by Horizon Europe Framework Programme of the European Union

KPI 3.3

User Feedback Modules have been collected and analyzed.

The test case produced was approached on Vemar's owned structures in continuity with what was shown in the project document and reported in Figure 1. For this reason, it was not necessary to prepare specific user models to analyze the feedback relating to the use of the system under examination.

KPI 3.4

The user guide is completed

The FLUIDOMOS user guide to setup the cluster infrastructure is available on the official project repository at https://github.com/vemarsas/FLUIDOMOS.

The user guide file is FLUIDOMOS/FLUIDOMOS Guide.md

KPI 3.5

Accounting records and documentation for expenditure reporting have been prepared and maintained for all the project activities.

Below the table listing all accounting records and documentation for expenditure for all the project activities.

Cost Category	Vemar sas	Costs	ICT Sistemi srl	Costs	Total amount
[redacted]	[redacted]	[redacted]			
	[redacted]	[redacted]			
	[redacted]	[redacted]			
			[redacted]	[redacted]	
			[redacted]	[redacted]	
					[redacted] €
[redacted]					

Funded by Horizon Europe Framework Programme of the European Union

[c] Equipment costs			Sensors	1.490,00 €	1.490,00 €
[d] Other direct costs	Consulting by Rosario Carvello	27.000,00 €			
			Consulting by Mauro Salvia	25.000,00 €	
					52.000,00 €
Total		60.000,00 €		58.560,65 €	118.560,65 €

Table 10: Accounting records for all the project activities

FLUIDOMOS Cluster Management

Proxmox Virtual Environment (VE) provides a robust platform for managing virtual machines (VMs), offering features such as snapshots and backups that are especially useful when deploying complex systems like Fluidomos. Fluidos lacks a built-in rollback mechanism for solver application, making it challenging to revert configuration errors or unintended changes. However, Proxmox's snapshot feature addresses this limitation by allowing administrators to create precise restoration points during the installation and configuration process.
Here's a detailed description of how Proxmox, through its snapshot capabilities, allows for precise rollback points during the installation and configuration of Fluidomos, bypassing the application's lack of a native rollback procedure for the solver. Additionally, Proxmox facilitates efficient management of virtual machine backups, enhancing system resilience and flexibility.
Snapshots in Proxmox capture the exact state of a VM, including memory, disk, and configuration. This functionality enables:

- Creating checkpoints at critical stages of Fluidomos installation or configuration.
- Experimenting with different setup parameters safely, knowing that any undesired changes can be reversed instantly.
- Ensuring a stable and reproducible environment for deploying Fluidomos across multiple instances.

By reverting to snapshots, administrators can efficiently resolve issues without requiring manual reinstallation or reconfiguration, saving significant time and effort.
In addition to snapshots, Proxmox supports scheduled and on-demand backups of virtual machines. This ensures that even in the event of hardware failure or data

Funded by Horizon Europe Framework Programme of the European Union

corruption, a complete VM, including its Fluidomos configuration, can be restored seamlessly. Backups complement snapshots by providing long-term storage and disaster recovery options.

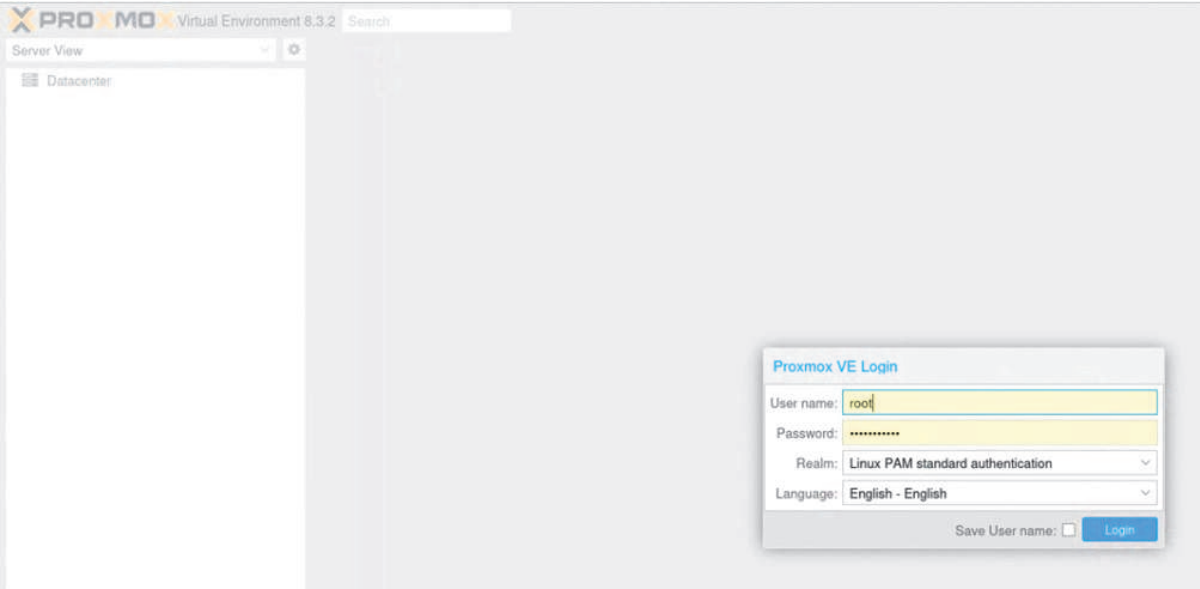


Figure 38 - Login Page

The Proxmox login interface provides secure access to the management environment, showcasing its user-friendly design.

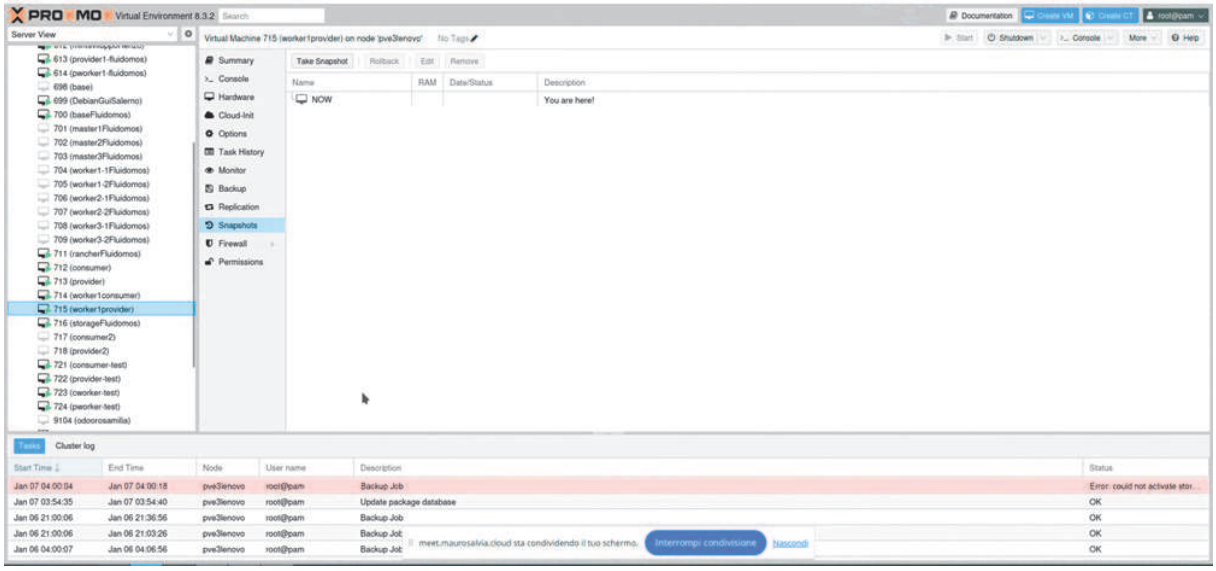


Figure 39 - Server View with Snapshots

The server view highlights the list of VMs with their snapshots, demonstrating the ability to manage and revert Fluidomos installations easily.

Funded by Horizon Europe Framework Programme of the European Union

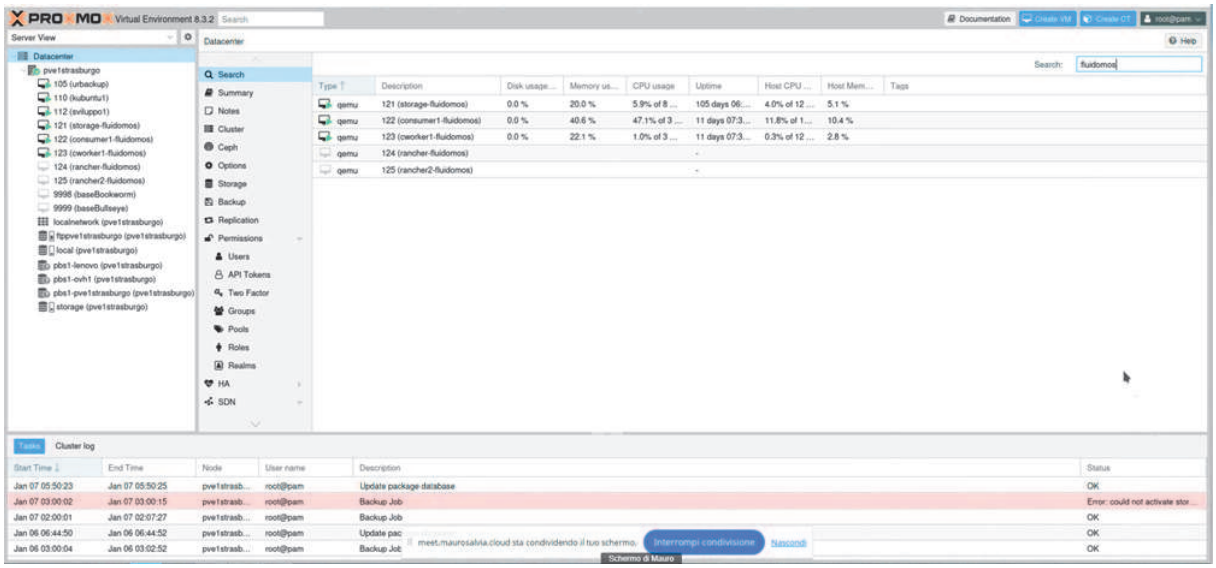


Figure 40 - Server View: Fluidomos V2 Consumer

Previous figure displays the VM hosting the Fluidomos consumer instance, allowing an overview of its status and configuration.

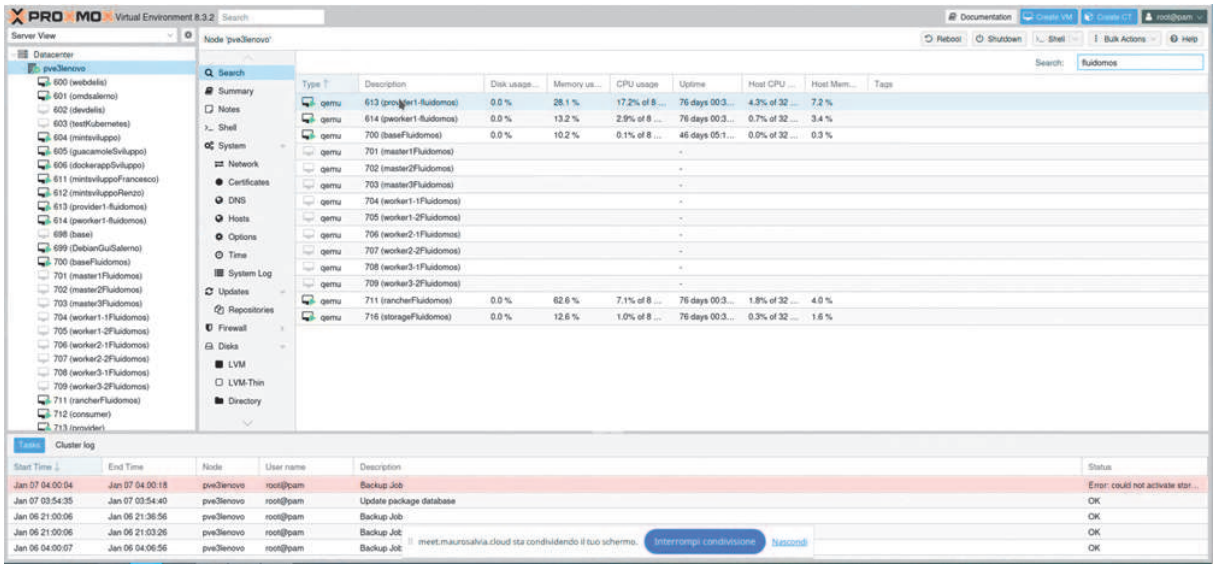


Figure 41 - Server View: Fluidomos V2 Provider

Above is highlighted the Fluidomos provider instance, showing its role in supporting consumer functionalities.

Funded by Horizon Europe Framework Programme of the European Union

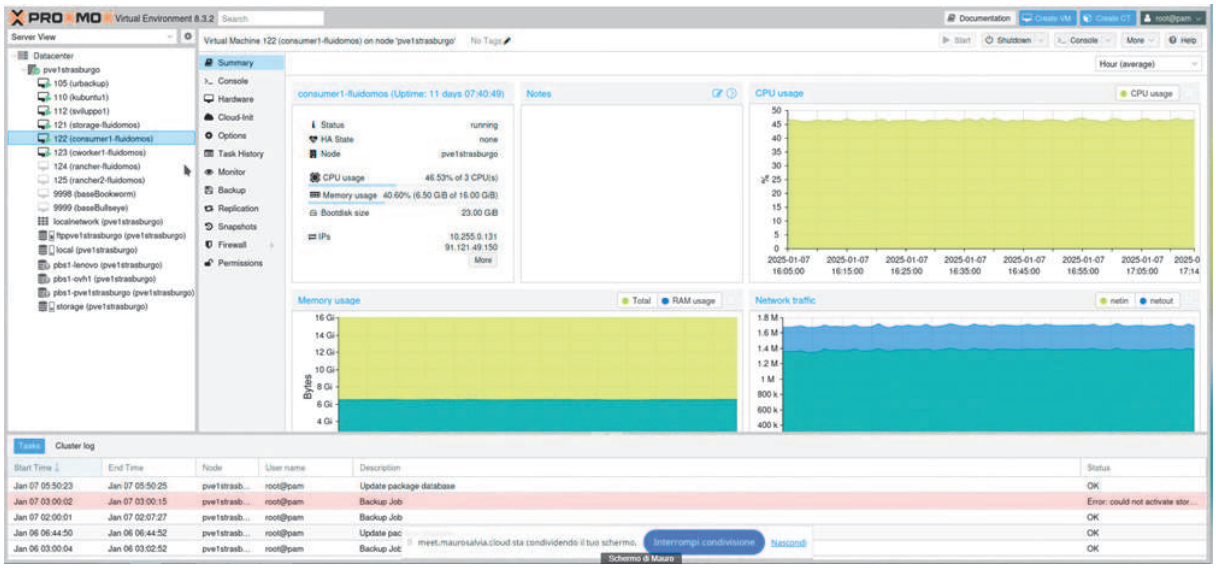


Figure 42 - Summary View of Control Plane Resources (Consumer)

Above figure provides an overview of CPU, memory, and disk usage for the consumer's control plane, aiding in resource monitoring and optimization.

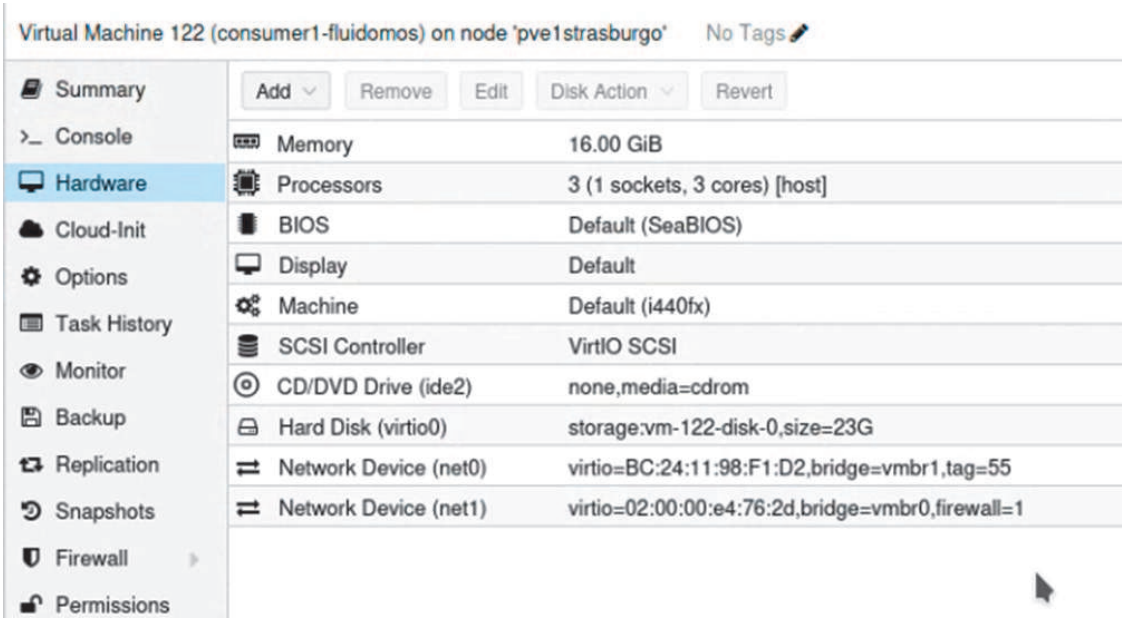


Figure 43 - Hardware Details of Control Plane (Consumer)

Hardware details dashboard shows the detailed hardware configuration (e.g., CPU cores, RAM, and storage) allocated to the consumer's control plane, ensuring proper resource allocation.

Funded by Horizon Europe Framework Programme of the European Union

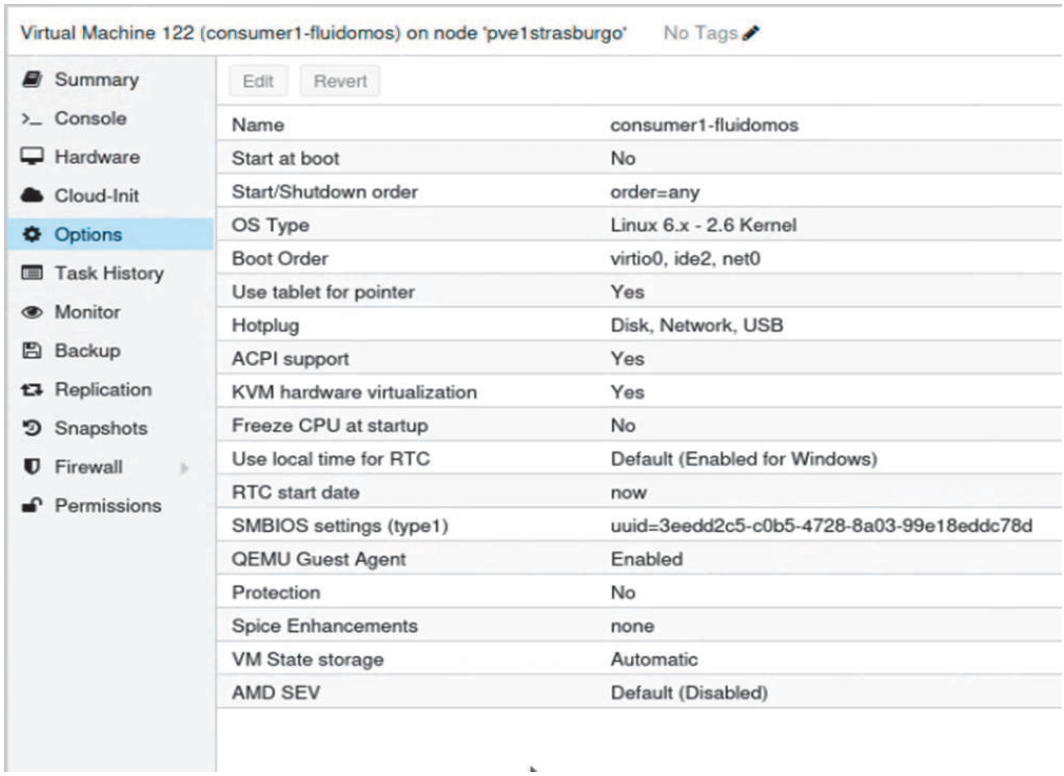


Figure 44 - Control Plane Configuration Options (Consumer)

Options section displays the configuration settings for the consumer's control plane, demonstrating how administrators can adjust VM parameters as needed.

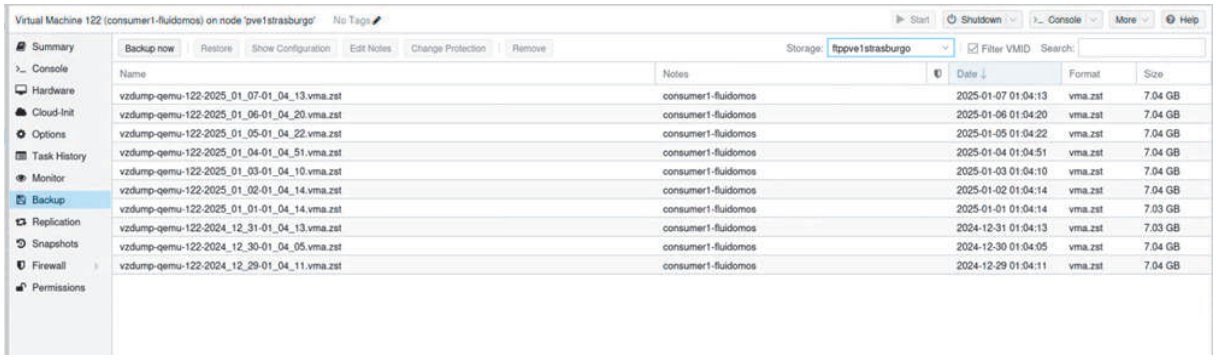


Figure 45 - Backups of Control Plane (Consumer)

Previously illustrated figure lists the available backups, emphasizing the importance of scheduled snapshots and backups in maintaining system integrity.

Funded by Horizon Europe Framework Programme of the European Union

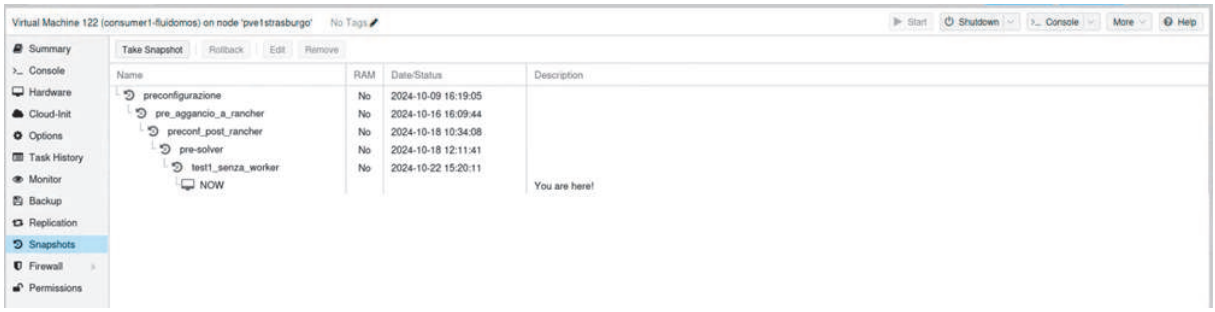


Figure 46 - Snapshots of Control Plane (Consumer)

The snapshot history for the control plane is shown above, demonstrating Proxmox's ability to create precise restoration points.

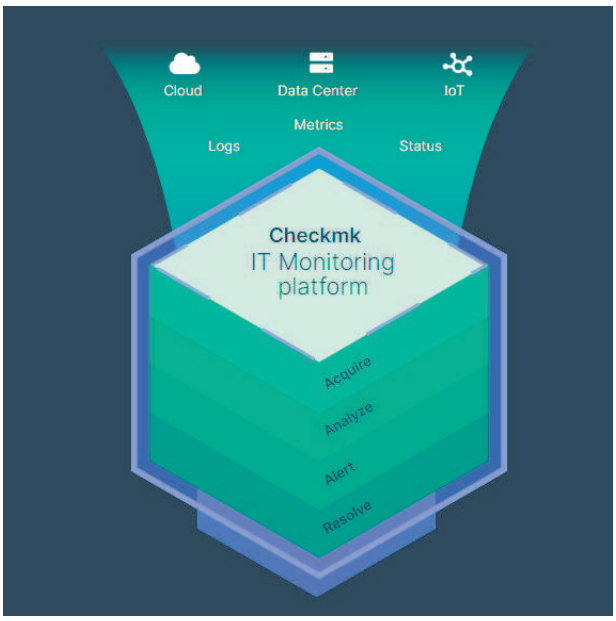
By leveraging Proxmox's snapshot and backup capabilities, administrators can effectively overcome the limitations of Fluidos's lack of rollback procedures. This approach ensures a stable, flexible, and recoverable environment, making Proxmox an ideal platform for deploying and managing complex applications like Fluidomos.

Funded by Horizon Europe Framework Programme of the European Union

Monitoring System

Metrics Monitoring Tool

The Checkmk tool was adopted to measure and analyze the performance of the systems under examination.



Checkmk is an open-source IT monitoring system developed in Python and C++. It is used to monitor servers, applications, networks, cloud infrastructures (public, private, and hybrid), containers, storage, databases, and environmental sensors. It includes various features such as State Monitoring (Records the "health" of a device or application through thresholds); Metric Monitoring (Allows recording and analysis of time-series graphs); Log and Event-Based Monitoring (Filters key events and triggers actions based on these events).

Some of the benefits that can be obtained with the use of this tool are related to Automation (Includes features for host auto-registration and lifecycle management), Scalability (Can monitor thousands of services with a single monitoring instance), API Integration (Integrates other systems using powerful APIs), Plug-ins (Offers over 2,000 actively maintained plug-ins to collect metrics from various IT systems).
Checkmk is designed to be scalable, automated, and extensible, providing a comprehensive view of IT infrastructure, from public cloud providers to data centres and is considered suitable for the purposes set within the scope of this project.

Funded by Horizon Europe Framework Programme of the European Union

Visualization Tool

Rancher provides a powerful platform for orchestrating Kubernetes clusters, simplifying deployment, monitoring, and scaling. For Fluidomos, Rancher enabled streamlined management of the **consumer control plane** and **worker nodes**, ensuring efficient resource allocation and application reliability.

Rancher served as a central monitoring tool for Kubernetes-based orchestration in the Fluidomos project. Its intuitive interface allowed for the deployment and scaling monitoring of the Fluidomos consumer control plane and worker nodes, as well as monitoring system health and performance. By leveraging Rancher, the team could efficiently monitor namespaces managing, workloads, and cluster resources, ensuring that Fluidomos operated reliably in a distributed environment.

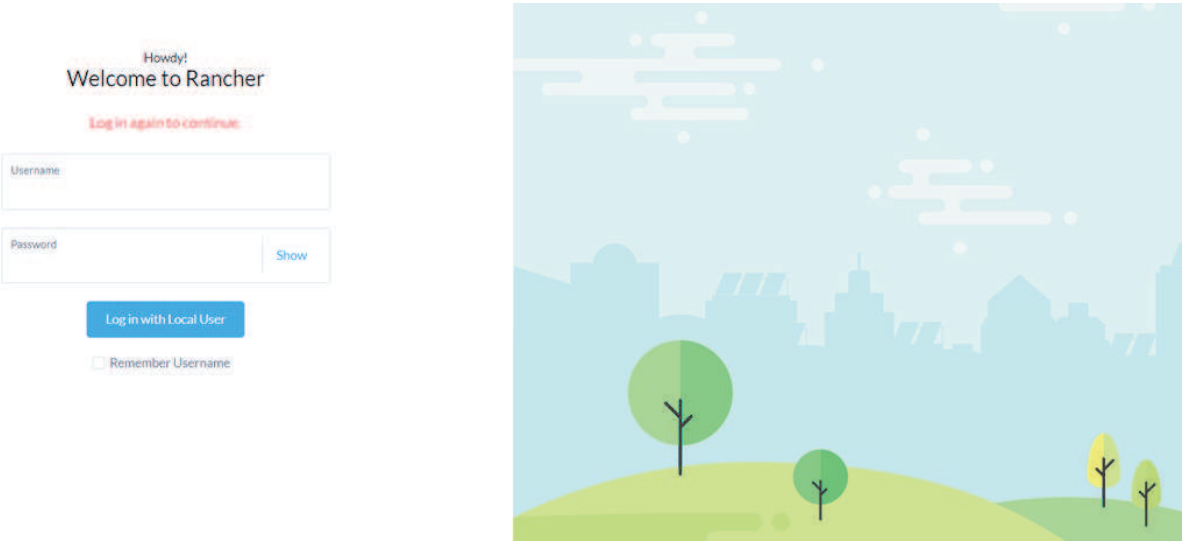


Figure 47 - Login Page

The Rancher login page showcases secure access to the platform, ensuring that only authorized users can manage the Kubernetes clusters and Fluidomos resources. Key features include single sign-on (SSO) support and multi-factor authentication (MFA).

Funded by Horizon Europe Framework Programme of the European Union

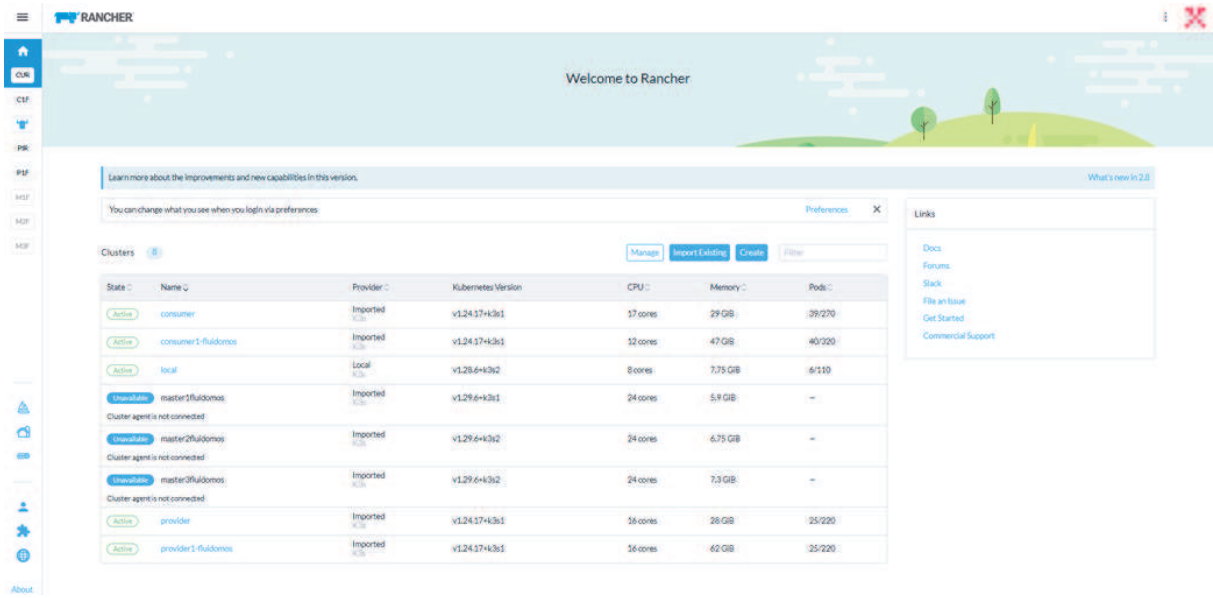


Figure 48 - Tool Dashboard

The dashboard provides a high-level overview of all Kubernetes clusters managed through Rancher. It includes cluster health, resource utilization, and access to detailed metrics for troubleshooting and monitoring.

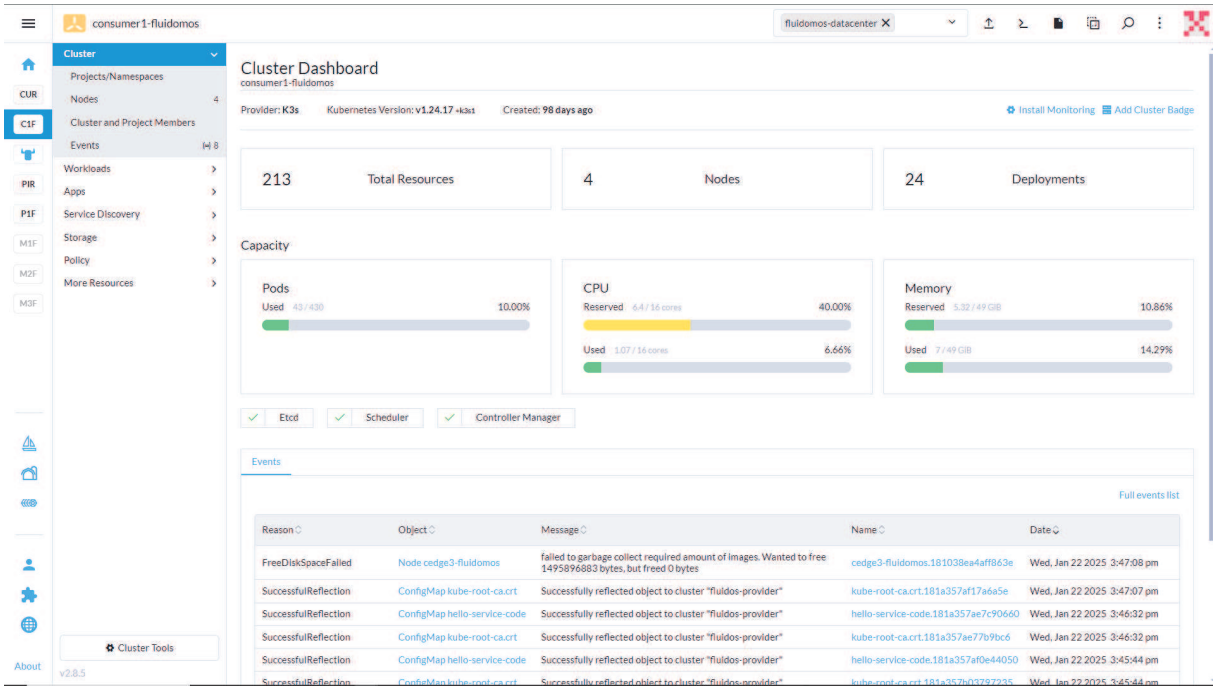


Figure 49 - Cluster Dashboard

Previous figure displays detailed information about a specific Kubernetes cluster hosting Fluidomos.
Key metrics include the number of nodes, running workloads, resource allocation, and system health indicators.

Funded by Horizon Europe Framework Programme of the European Union

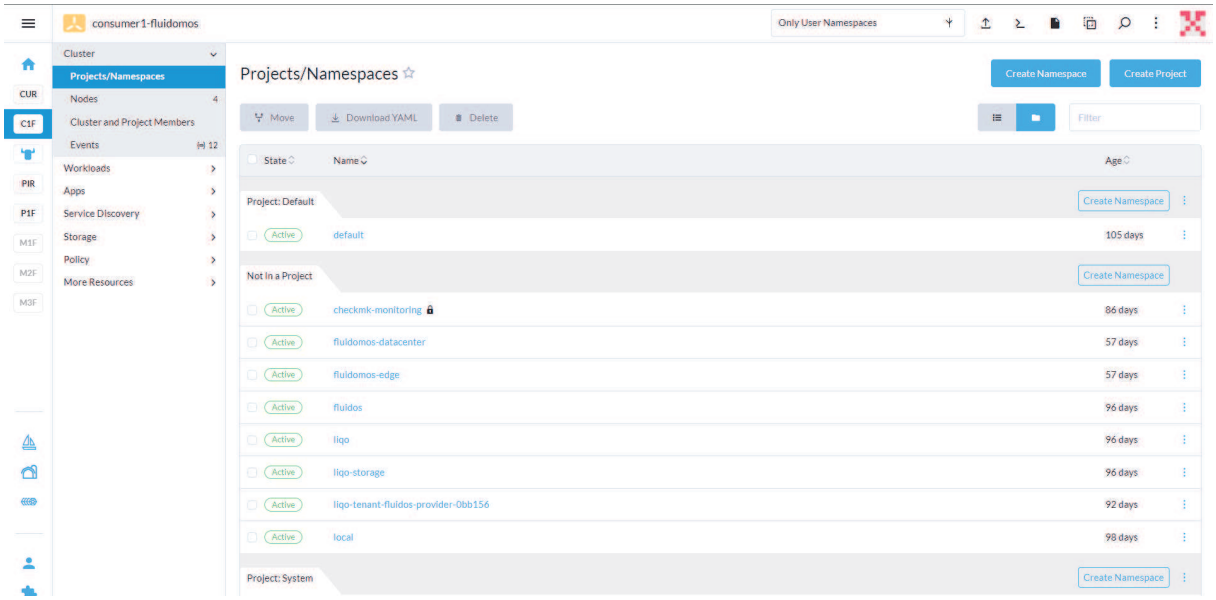


Figure 50 - Project Namespaces

Projects and namespaces are used to organize and isolate workloads within the cluster. Fluidomos consumer and provider instances are assigned to dedicated namespaces to ensure proper resource segmentation and access control.

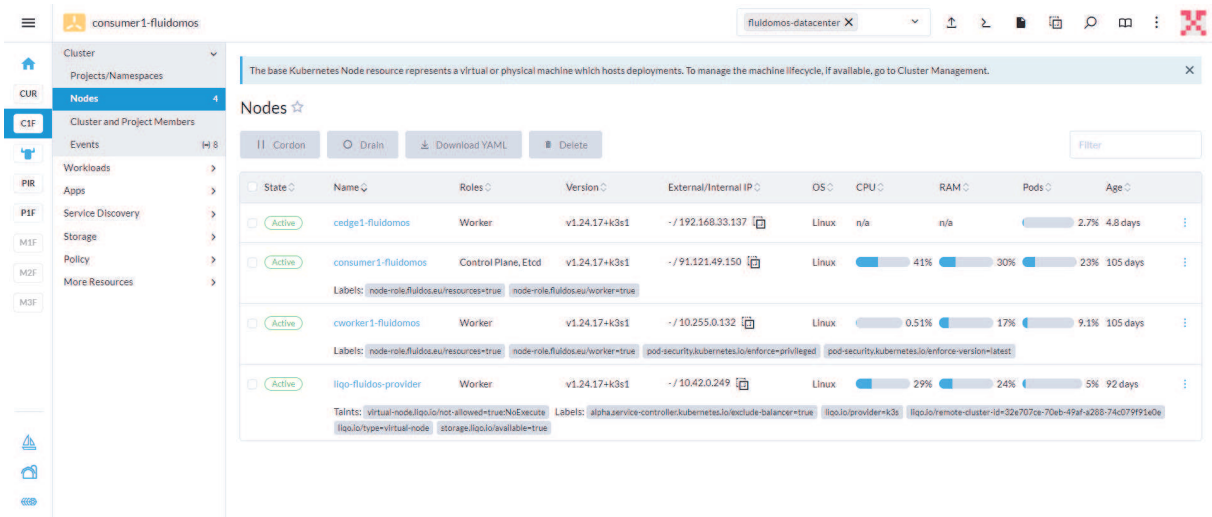


Figure 51 – Nodes

The nodes section lists all worker nodes in the cluster, displaying their resource usage (CPU, memory, and storage). This is related to the Fluidomos cluster consumer. For Fluidomos, this helps identify any potential bottlenecks or nodes requiring scaling.

Funded by Horizon Europe Framework Programme of the European Union

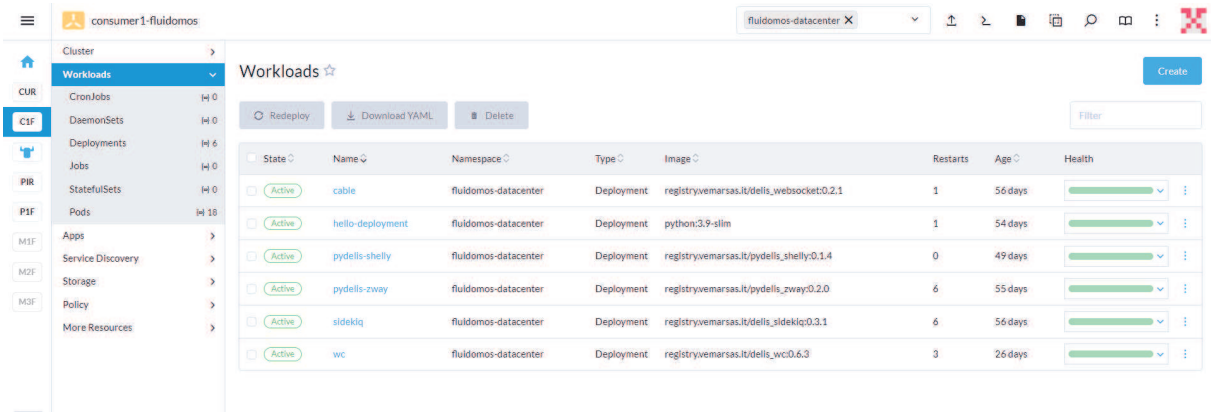


Figure 52 - Workloads

Previous section provides a detailed view of all running workloads in the cluster, including deployments, pods, and stateful sets. Administrators can monitor the status of each workload, such as the Fluidomos control plane and worker components.

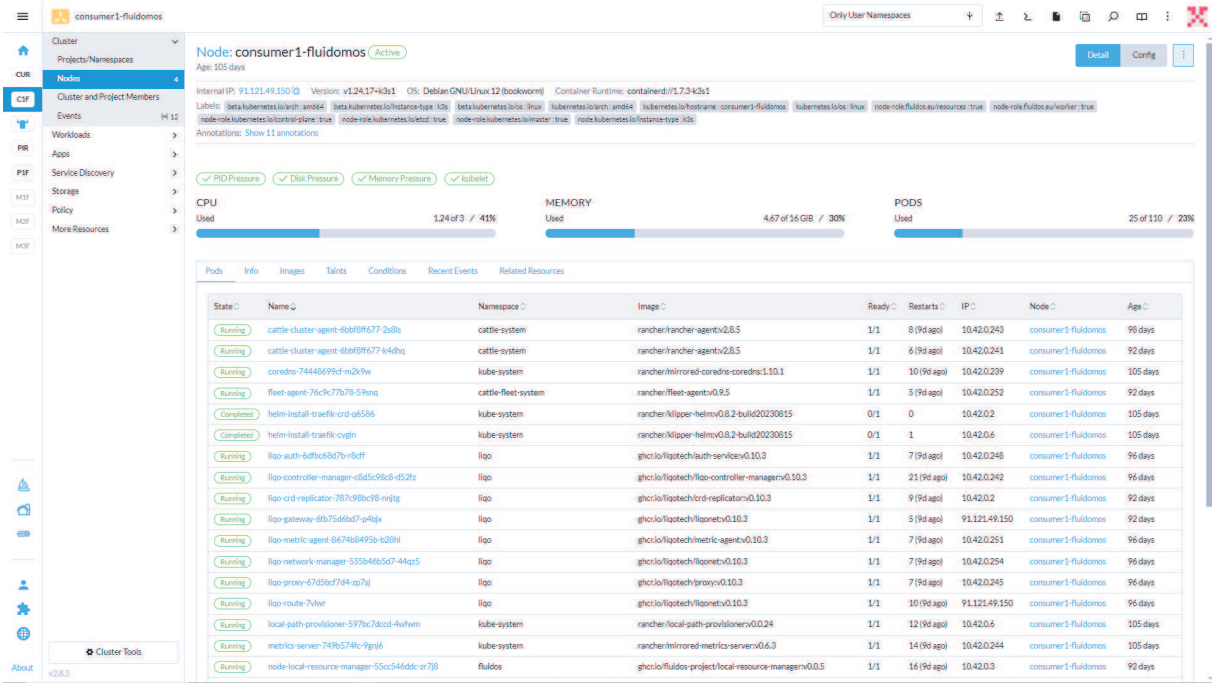


Figure 53 - Fluidomos consumer control plane

This workload represents the core management layer of the Fluidomos consumer. Rancher allows administrators to monitor the control plane's resource usage, logs, and health checks, ensuring smooth operations.

Funded by Horizon Europe Framework Programme of the European Union

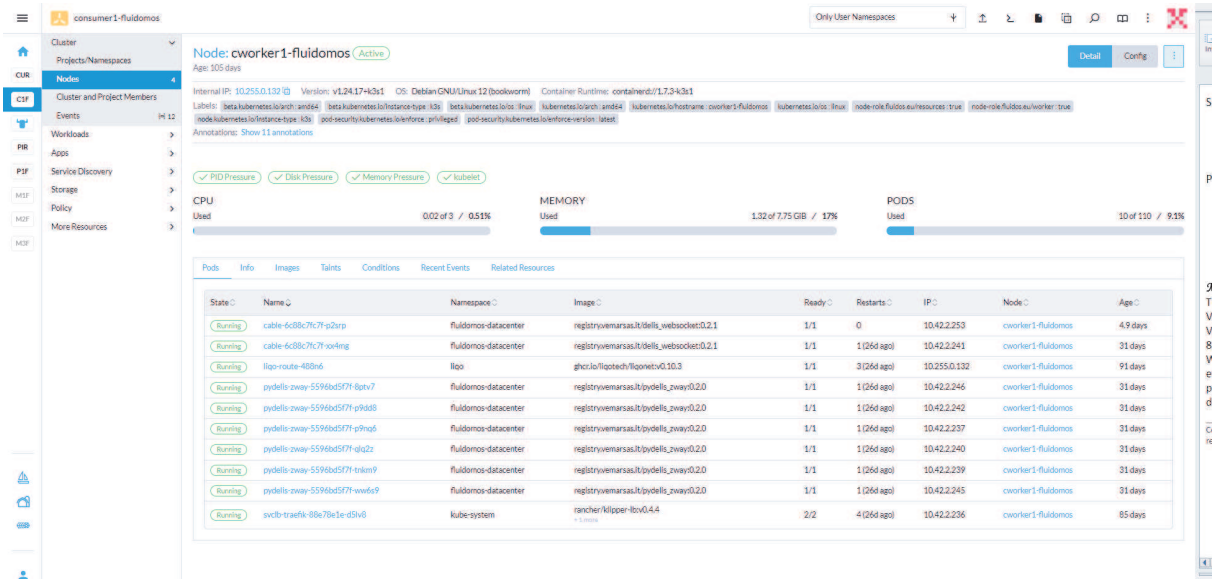


Figure 54 - Fluidomos consumer worker

Above figure represents the worker nodes responsible for executing Fluidomos computational tasks. Rancher provides insight into pod scheduling, workload distribution, and scaling options for these nodes.

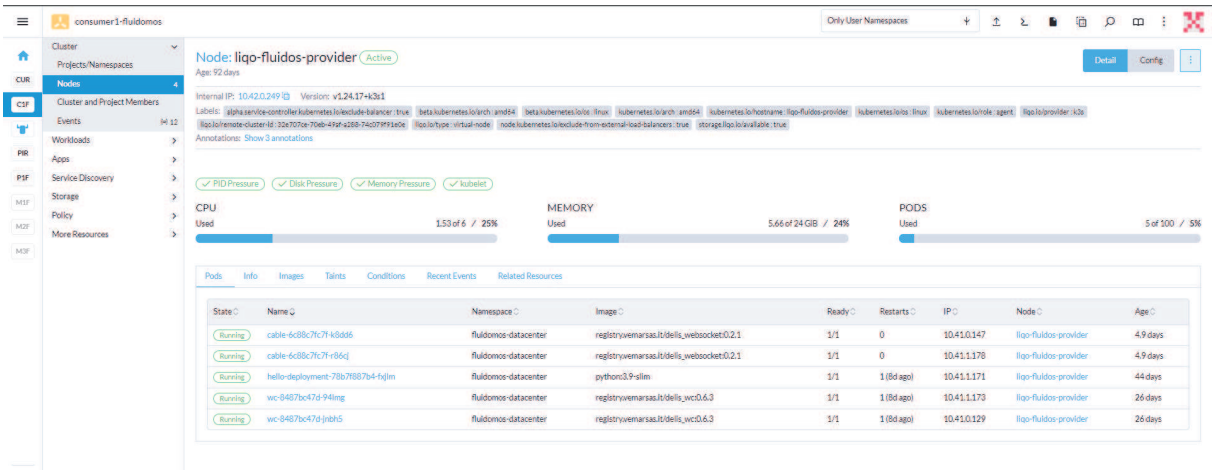


Figure 55 - fluidomos consumer liqo-fluidos-node

This figure illustrates the Liqo node running the Fluidomos provider within the Rancher-managed Kubernetes cluster. Liqo, a solution for multi-cluster orchestration, enables seamless resource sharing across different clusters. In the context of Fluidomos, the Liqo node facilitates distributed workloads by acting as a provider of computational resources.

Node Information:The figure shows the provider node's name, status, and role within the Kubernetes cluster.Examples of roles include "worker" or "infrastructure" specific to resource provisioning.

Resource Metrics:Real-time usage of CPU, memory, and storage by the provider node is displayed.These metrics help ensure that the Liqo node is operating efficiently and not overburdened.

Funded by Horizon Europe Framework Programme of the European Union

Workloads on the Node:Lists the Fluidomos provider-specific workloads scheduled on the node. Includes pods, their status (Running, Pending, etc.), and any network integrations specific to Liqo.
Health Status and Logs:The figure provides quick access to node health indicators, such as readiness checks and error logs. This allows for proactive troubleshooting if issues arise with the Fluidomos provider's node operations.

The provider Liqo node role in Fluidomos Workflow is critical for extending the computational capacity of the Fluidomos system by enabling multi-cluster resource allocation. Rancher facilitates:

- Monitoring the node's performance and ensuring it meets the demands of distributed workloads.
- Simplifying the management of Liqo configurations, such as cluster peering and workload balancing.
- Supporting scalability by adding or removing provider nodes as needed.

Using Rancher for Fluidomos we simplified Monitoring and Troubleshooting. The dashboard and workload views enabled real-time monitoring of cluster health and quick identification of issues.

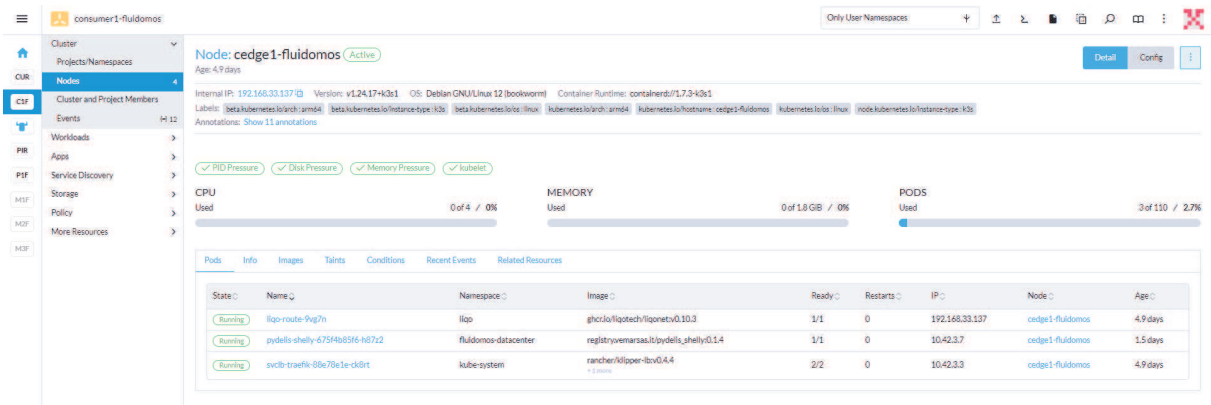


Figure 56 - Fluidomos consumer edge node

The figure represents the node status view of a Kubernetes cluster, specifically for the node cedge1-fluidomos, which is active and has been operational for a while. The information is organized into key metrics and statuses related to CPU, memory, and pods usage, along with details of running pods.

The node has very low CPU, memory, and pod utilization, leaving ample capacity for additional workloads or scaling needs. All pods are running without critical issues. Minimal restarts indicate good stability and reliability. The absence of pressure on CPU, memory, or disk resources demonstrates a well-maintained and lightly utilized environment.

The node cedge1-fluidomos is in excellent health, with stable performance, no signs of resource pressure, and minimal pod issues. However, it is still underutilized, and some minor adjustments could improve resource efficiency and system performance.

Funded by Horizon Europe Framework Programme of the European Union

4. Benefits

According to the project documents, the benefits resulting from the activities carried out on Fluidomos are various.
The more or less visible ones will be highlighted below, also in relation to what was hypothesized in the design phase.

The innovation potential

Considering home or business scenarios, FLUIDOMOS will leverage the principle of liquid computing to seamlessly make available resources and services in networks implemented with edge-cloud architecture. In such scenarios, where current applications are deployed with on-premise or cloud solutions, this approach is an important advancement that will bring measurable benefits in terms of reduced latency, improved privacy, resource sharing, and business continuity while trying, at the same time, to decrease the use of energy resources needed to perform human activities.

The circular nature of the project

FLUIDOMOS will contribute to environmental sustainability for two reasons. The first refers to the inherent characteristics of home automation systems, such as DELIS, which can be configured and used for energy-saving purposes. In this respect, newly established energy communities will benefit from smart energy management. The second, derivable from FLUIDOS, is to reduce network energy consumption by minimizing the amount of data transmitted over long distances. By limiting the dependence on far-flung data centers and using local processing when possible, Edge Computing contributes to more sustainable and efficient network operations.
aggiungere la parte sotto di marco sul rispa

Energy Efficiency

Fluidomos likely uses dynamic workload balancing to minimize idle energy consumption across nodes. Supports power-efficient hardware and advanced scheduling, leading to reduced overall power usage. Since the infrastructure is already in place, deploying the machines on it has a minimal impact on the energy consumption which it supports, certainly lower than the extra energy consumption that would be required on external VPS.

Funded by Horizon Europe Framework Programme of the European Union

Economic Savings

Streamlined infrastructure reduces maintenance and operational costs. Centralized or optimized resource allocation lowers costs related to hardware procurement and scaling. The economic savings are related also to the fact that, since you already have an IT infrastructure in a data center you own, the deployment of a cluster inside it is free of charge so are related to the avoiding of cost to purchase and maintain a provider and related workers in a third-party provider.

Quality and Feasibility of the Technical Approach

FLUIDOMOS will redesign the current DELIS interfaces to Controllers and IOT devices as Cloud-Native applications using loosely coupled microservices running in containers managed by K8S and FLUIDOS. The quality of the proposed technical approach will derive from the adoption of Kubernetes patterns that guarantee the reuse of certified solutions and the reduction of development time. Additionally, each containerized microservice will cover both build time and runtime design principles, such as:

- Single Concern: Each container addresses a single concern and does it well.
- Image Immutability: Containerized microservices are meant to be immutable.
- High Observability: Every container must implement all necessary APIs to help to observe and manage the IOT device in the best way possible.
- Process Disposability: Containerized services must be as ephemeral as possible and ready to be replaced by another container instance at any point in time.
- Runtime Confinement: Every container must declare its resource requirements and restrict resource use to the requirements indicated.

Soundness of the Approach and Credibility.

The proposed approach is aligned with the state of the art of research in IoT and liquid computing. In particular, the principles and techniques described in the following papers will be used for the design and implementation of FLUIDOMOS:
1) Qiu, T., Chi, J., Zhou, X., Ning, Z., Atiquzzaman, M., & Wu, D. O. (2020). Edge computing in industrial internet of things: Architecture, advances and challenges. IEEE Communications Surveys & Tutorials, 22(4), 2462-2488.
2) Iorio, M., Risso, F., Palesandro, A., Camiciotti, L., & Manzalini, A. (2022). Computing without borders: The way towards liquid computing. IEEE Transactions on Cloud Computing.

Funded by Horizon Europe Framework Programme of the European Union

Resilience

Fluidomos could leverage distributed architecture and improved fault tolerance mechanisms, ensuring the system remains operational despite localized failures in nodes or services. It allows better redundancy and dynamic failover capabilities.

The use of FLUIDOS stimulates to solve some critical issues of DELIS and provides optimization mechanisms for distributing computing resources. The following figure highlights these aspects by comparing a canonical usage scenario of DELIS via the Cloud (on the left side) with an alternative one (right side), which may be possible through FLUIDOS in the following circumstances:

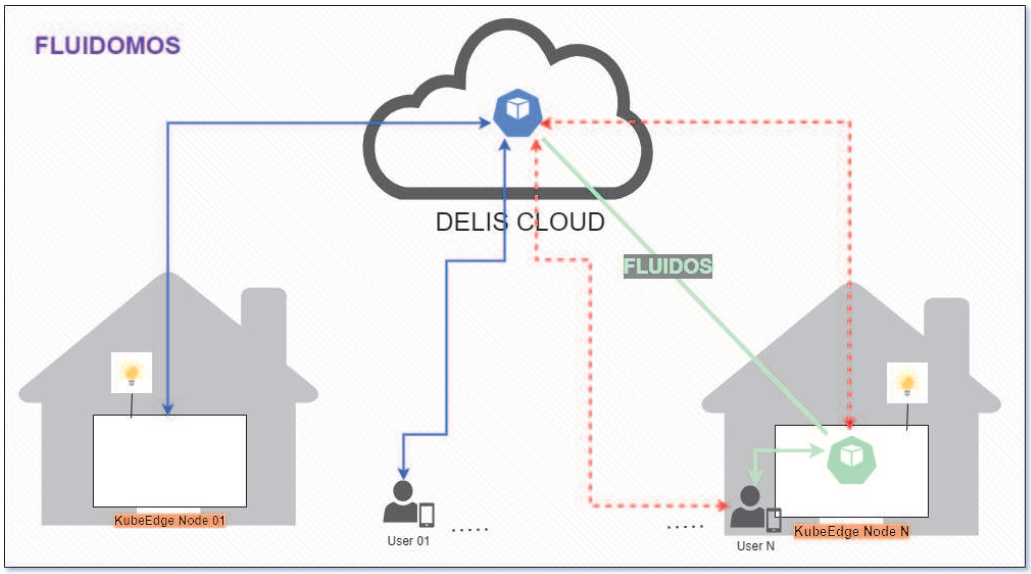


Figure 57: The Impact of FLUIDOS on FLUIDOMOS

1. Unavailability of communication with the Cloud: In this case, the benefits of service continuity are obtained.
2. The user's proximity to their own home automation system. In this scenario, access to the Cloud, which is no longer necessary, could be interrupted, obtaining the benefit of a better distribution of the computational workload between the Cloud and Edge.
3. Need to bind the access to services and data into a specific region: For example, a generic maintenance technician, authorized to maintain all home automation systems distributed along a set of buildings, for any security restrictions needs, could be bound to access only to the services of a single node, rather than to global services running in the Cloud.

In a DELIS-based architecture integrated with Fluidomos, the reliance on a centralized PostgreSQL database in the cloud can lead to operational interruptions when edge nodes lose connectivity to the database. A practical solution to this challenge is the implementation of logical replication of database tables on edge devices. By maintaining a localized replica of the relevant data tables and enabling bidirectional synchronization, edge nodes can continue functioning independently

Funded by Horizon Europe Framework Programme of the European Union

during disconnections and seamlessly resynchronize when the connection is restored.

Implementation

Edge Database Replication Setup - A lightweight PostgreSQL instance is deployed on each edge node. Logical replication is configured to replicate only the data relevant to each edge node's operations from the central database in the Cloud.

Bidirectional Synchronization - Changes made locally on the edge node (e.g., data updates, new entries) are queued and synchronized with the central database once connectivity is re-established. Updates from the central database are periodically propagated to the edge replica when a connection is available.

Conflict Resolution - A conflict resolution strategy is under evaluation to handle simultaneous updates on both the edge and central database. The ideas are related to

- Timestamp-based resolution: Prioritize the most recent update.
- Predefined rules: Apply specific business logic to decide which update takes precedence.
- Data Consistency - Mechanisms like change data capture (CDC) ensure consistency by tracking all modifications and applying them in the correct order during synchronization.

Benefits and Advantages

Resilience to Connectivity Loss - Edge nodes remain operational during network outages, reducing downtime and ensuring continuity of critical processes.
Improved Latency - By querying and writing data locally, edge nodes experience lower latency compared to accessing a remote database in the cloud.
Reduced Bandwidth Usage - Frequent reads and writes are handled locally, minimizing the volume of data transmitted over the network.
Scalability - The system can scale to multiple edge nodes, each maintaining its own replica of the relevant data, without overwhelming the central database.
Data Integrity and Synchronization - Logical replication ensures that all changes are tracked and reconciled, maintaining data integrity across edge and central nodes.
Flexibility - Custom replication filters can be applied to replicate only the data required for each edge node, optimizing storage and processing resources.
Seamless Integration with Fluidomos - Fluidomos leverages the edge replicas for uninterrupted access to data, enabling consistent operation and decision-making even during cloud connectivity issues.

Implementing PostgreSQL logical replication for edge data in a Fluidomos environment enhances the robustness and efficiency of DELIS-based microservices. By enabling edge nodes to function autonomously and synchronizing data bidirectionally with the central database, this approach ensures system resilience,

Funded by Horizon Europe Framework Programme of the European Union

reduces latency, and optimizes resource usage, offering a scalable solution for distributed industrial applications.

Optimized Edge Node Utilization

Improved orchestration ensures edge nodes handle workloads intelligently, reducing latency and avoiding overloading specific nodes. Enables edge computing closer to the data source, improving efficiency and reducing backhaul traffic to central servers. Moreover, given the limited computing capabilities of the edges, having the possibility of distributing service replicas (pods) on the various nodes, we can fully exploit the capabilities of the edge without overloading it.

Conclusion

As a tangible result, FLUIDOMOS will provide a computing continuum in the domotic ecosystem, spanning multiple technological domains (cloud, edge, smart IOT devices) and multiple administrative boundaries (e.g., home users, maintainers, providers).

During the project's closing phase, it is clear that FLUIDOMOS has successfully advanced in the direction outlined during the initial proposal phase, achieving nearly all the objectives that were set.

One of the key challenges identified was the integration of a plurality of technologies and protocols into interoperable services for home/building automation applications, and the orchestration of these services. FLUIDOMOS has effectively addressed this challenge by implementing a meta OS that facilitates seamless integration of heterogeneous hardware and software. Moreover, through the mechanisms provided by FLUIDOS, the project overcame the traditional limitations of centralized systems (e.g., dependency on a single cloud server). By enabling horizontal and vertical interaction mechanisms, FLUIDOMOS has increased system resilience, reduced latency, and ensured continuity of operations even in scenarios with limited or no connectivity.

In conclusion, FLUIDOMOS has delivered a robust framework that addresses the growing complexity of home/building automation systems by enabling seamless interoperability, enhanced resilience, and a more secure service model. These achievements mark substantial progress toward the project's goals and demonstrate the practical applicability of the proposed solutions in real-world scenarios.

© 2022-2025 FLUIDOS

Funded by Horizon Europe Framework Programme of the European Union