



Cognitive edge-cloud with serverless computing

D5.1: Use case definition and initial platform integration

Due date of deliverable: 31/03/2024

Actual submission date: 12/04/2024



This project has received funding from the EUROPEAN HEALTH AND DIGITAL EXECUTIVE AGENCY (HADEA) program under Grant Agreement No 101092950

Disclaimer: This document reflects only the author's view and the European Commission is not responsible for any use that may be made of the information it contains. This material is the copyright of EDGELESS consortium parties, and may not be reproduced or copied without permission. The commercial use of any information contained in this document may require a license from the proprietor of that information.

The EDGELESS Consortium

Legal Name	Role	Country
WORLDLINE IBERIA SA	COO	ES
TECHNICAL UNIVERSITY OF CRETE	BEN	EL
CONSIGLIO NAZIONALE DELLE RICERCHE	BEN	IT
THALES SIX GTS FRANCE SAS	BEN	FR
UBIWHERE LDA	BEN	PT
SIEMENS AKTIENGESELLSCHAFT	BEN	DE
TELEFONICA INVESTIGACION Y DESARROLLO SA	BEN	ES
FUNDACIO EURECAT	BEN	ES
TECHNISCHE UNIVERSITAET MUENCHEN	BEN	DE
INFINEON TECHNOLOGIES AG	BEN	DE
AEGIS IT RESEARCH GMBH	BEN	DE
THE CHANCELLOR MASTERS AND SCHOLARS OF THE UNIVERSITY OF CAMBRIDGE	AP	UK



Document Information

Contractual Date of Delivery	31/03/2024
Actual Date of Delivery	12/04/2024
Deliverable Security Class	Public
Editor	<i>Eva Rodrigues (UW), Rita Santiago (UW), Hélio Simeão (UW)</i>
Contributors	<i>Eva Rodrigues (UW), Rita Santiago (UW), Francisco Vicente (WLI), Mercedes Verhaz (WLI), Jesus Omaña (TID), Eric Domingo (EUT), Nestor Garcia (EUT), Carlos Molina (EUT), Panagiotis Antoniou (AEGIS), Chronis Ballas (AEGIS)</i>
Quality Assurance	<i>Markus Sauer (SIEMENS), Roman Kolcun (CAM)</i>

Revisions

Version	Date	Revision Author	Summary of Changes
0.1	05.03.2024	Eva Rodrigues, Rita Santiago,	Initial Complete Version
0.2	23.03.2024	Markus Sauer, Roman Kolcun	Internal Review
0.3	05.04.2024	Hélio Simeão, Eric Domingo, Fan Vicente	addressing Internal Review comments
1.0	12.04.2024	Hélio Simeão	Final Version



Executive Summary

This deliverable reflects the work conducted during the first ten (10) months of WP5 Integration & validation. The Platform Integration section refers to the activities in the scope of the Task 5.1 (T5.1) Prototype integration and testing platform, where the Continuous Integration and Continuous Deployment (CI/CD) approach, tools and processes are described. The following sections of this document refer to the remaining tasks of WP5, namely the three (3) project Use Cases (UCs), Smart City Analytics, Internet of Robotic Things and HealthCare Assistant, T5.2, T5.3 and T5.4 respectively.

In each of the UC specific sections, a description of the UC, its objectives and the target hardware used are described. The main motivations and challenges, UC specific requirements and its translation into some of the edgeless features are depicted, as well as an Edgeless Workflow for each UC that results from cross-WP collaboration efforts. Lastly, the UCs also provide a macro roadmap, open points and next steps to implement the desired edgeless features, defining an initial proposal on how to test and validate them.



Table of Contents

Executive Summary.....	4
List of Figures	6
List of Tables	6
1 Introduction	7
2 Platform Integration	8
2.1 Context.....	8
2.2 Methodologies, Tools and Workflows	8
2.3 EDGELESS Integrated Framework	10
2.4 Edgeless Open Source Repository.....	13
2.5 CI/CD Processes	15
3 Use cases.....	16
3.1 Autonomous Smart City Surveillance (ASCS)	16
3.1.1 Description	16
3.1.2 Motivation & Challenges.....	19
3.1.3 Edgeless Workflow	21
3.1.4 Open points & next steps.....	22
3.2 Internet of Robotic Things (IoRT).....	23
3.2.1 Description	23
3.2.2 Motivation & Challenges.....	25
3.2.3 Edgeless Workflow	26
3.2.4 Open Points and Next Steps.....	28
3.3 HealthCare Assistance (HCA)	29
3.3.1 Description	29
3.3.2 Motivation & Challenges.....	34
3.3.3 EDGELESS Workflow.....	36
3.3.4 Open points and Next Steps.....	38
4 Conclusion.....	40
6 References	41

List of Figures

Figure 1: ASCS Real Scenario Overview	17
Figure 2: Smart Lamp Post example	18
Figure 3: NVIDIA Jetson Xavier and ANNKE 1080p 1920TVL Security CCTV Bullet Camera	19
Figure 4: Autonomous Smart City Surveillance Use Case architecture	22
Figure 5: IoRT UC Architecture diagram integrated with EDGELESS workflows	26
Figure 6: Person's home with sensors depicted	30
Figure 7: Malaga Platform (Victoria Network).....	32
Figure 8: Data Center Architectures	33
Figure 9: UC Architecture diagram integrated with EDGELESS workflows	36
Figure 10: UC Interconnection diagram with EDGELESS.....	38

List of Tables

Table 1: Overview of EDGELESS system components and integration process	11
Table 2: EDGELESS project main repository structure.....	13
Table 3: Key hardware sensors planned for the Internet of Robotic Things Application.	24
Table 4: Overview of the sensor types.....	30



1 Introduction

EDGELESS intends to leverage the serverless concept in all the layers in the edge-cloud continuum to fully benefit from diverse and decentralised computational resources available on-demand close to where data are produced or consumed. This forward-thinking initiative emerges from the seamless integration of diverse elements into an advanced system, as elaborated in the initial version documented in D2.1, titled "Initial requirements, programming model, and architecture."

To bring this integrated system to fruition, EDGELESS embraces a Continuous Integration/Continuous Delivery (CI/CD) approach, a fundamental tenet of the DevOps methodology. This methodology ensures streamlined and efficient software development, effectively addressing bottlenecks. The project employs established methodologies, including regular online meetings and iterative development cycles facilitated through communication on Slack. GitHub is the primary platform for software lifecycle and DevOps activities, and an Edgeless GitHub project¹ was created, hosting a set of repositories related to the project, namely the main Edgeless repository², fostering a structured and efficient collaborative environment. The Platform Integration section intricately elucidates EDGELESS's methodology for integration and delivery, offering insights into the employed methods. The project's utilisation of GitHub Actions for CI workflows is underscored, showcasing its pivotal role in critical operations triggered by specific actions within repository branches.

The narrative seamlessly transitions to in-depth explorations of three significant use cases within the EDGELESS project, each illustrating the framework's applicability in real scenarios and which features are required from Edgeless:

Autonomous Smart City Surveillance Use Case examines sub-cases like entire edge and hybrid edge-cloud scenarios. The integration of EDGELESS in edge nodes with varied hardware capabilities is showcased through a comprehensive analysis of a smart city scenario involving CCTV cameras strategically positioned on smart lampposts.

The Internet of Robotic Things use case highlights the integration of autonomous robotics and IoT-connected sensors in agile manufacturing scenarios. EDGELESS integration enables orchestrating robotic teams and dynamically assigning tasks based on real-time data from IoT sensors, and optimises manufacturing resources, emphasising their adaptability and efficiency in industrial environments.

The Healthcare Assistance Use Case focuses on designing and developing a personal assistant with activity identification and anomaly detection capabilities. The system leverages IoT devices in the homes of elderly individuals, showcasing the advantages of running detection algorithms on edge devices for low-latency processing, enhanced privacy and using the stateful functions provided by the EDGELESS system, allowing seamless monitoring of daily activities.

¹ <https://github.com/edgeless-project>

² <https://github.com/edgeless-project/edgeless>



2 Platform Integration

2.1 Context

EDGELESS introduces a groundbreaking framework for serverless edge computing, this section delves into the seamless integration of the system and its components.

This vision stems from linking different heterogeneous elements, comprising components and features sourced from a diverse range of partners and potentially third-party contributors, into a complex system based on a customised architecture, whose preliminary version is reported in D2.1, “Initial requirements, programming model, and architecture”.

To achieve the integrated system, EDGELESS will use a Continuous Integration/Continuous Delivery (CI/CD) approach (i.e., integrating code into a shared repository at regular intervals and subsequent testing of the integrated). CI/CD is employed in the DevOps (Development and Operations) approach followed in EDGELESS to minimise bottlenecks and make software building, testing, and release more efficient and lean.

This section will describe the generic framework for integration and delivery followed in EDGELESS.

2.2 Methodologies, Tools and Workflows

EDGELESS adopts a set of common methodologies to ensure consistency and efficiency across all teams. This includes regular online meetings and iterative development cycles. We utilise Slack as our primary communication platform, with a dedicated channel within Slack specifically for integration-related discussions. These methodologies facilitate effective communication and timely adjustments to the project plan.

The following tools have been elected to organise the cross-partner development and integration team. EDGELESS will use GitHub as its software lifecycle and DevOps platform, incorporating many features to handle the end-to-end software development and operations workflows. In this respect, EDGELESS will utilise the following GitHub features:

- **GitHub Source Code Management.** GitHub will serve as a centralised source code repository and management system to keep track of every part of the EDGELESS source code contributed by all partners. Using GitHub as a distributed version control system tool, developers can clone the code on a local machine, creating a safe environment for experimentation without affecting the overall system's functionality. Once the desired features or changes have been implemented, the updated code is pushed back to the remote repository, making it available to the entirety of the development team. This enables several developers to work on the same source code base without affecting one another, while the amount of effort required to handle collisions during a pull request minimises. A dedicated EDGELESS Organization has been set up on github.com with a Team for each partner. Repositories are available under the Organization.



- **GitHub Actions.** GitHub incorporates tools for continuous software development methodologies that enable building and testing the software as soon as the developer commits a file to the repository (CI), and provides a production build of the CI-validated code at any given time (CD).
- **GitHub Issues.** GitHub Issues is an essential issue-tracking tool for managing tasks, bugs, enhancements, and other relevant requests in the EDGELESS project. It allows for effective task assignment and progress tracking through tagging and direct linking to relevant source code. This platform fosters clear, direct communication among team members, providing a space for detailed discussions on each issue, leading to more efficient and collaborative problem resolution.
- **GitHub Releases.** GitHub Releases is a powerful tool that will be used for managing software releases in the EDGELESS project. It integrates seamlessly with GitHub's version control, enabling developers to tag and release specific source code versions. With GitHub Releases, the EDGELESS team can easily publish software packages, release notes, and binaries, creating a clear and accessible version history.
- **GitHub Wiki.** Using a documentation platform reminiscent of Wikipedia, GitHub Wiki is a tool for maintaining documentation for a project. While anyone can view a wiki page, only a designated developer can create and edit its contents.

In the EDGELESS project, we have established a central repository, aptly named [edgeless](#), which is the heart of our development efforts. This primary repository houses the core components pivotal to our project's success. Alongside this, several other repositories are dedicated to various other components and aspects of the project. By structuring our project this way, we maintain clarity and efficiency, allowing each team to contribute effectively to their respective areas while keeping the central repository as the convergence point for our major developmental strides.

Methodology for Collaborative Work

Our approach to managing collaborative contributions from developers across different partner teams in the EDGELESS project is meticulously structured to ensure efficiency and maintain the integrity of our main repository. The methodology we follow is outlined as follows:

- **Pre-disclosure of Contributions.** Developers are encouraged to disclose their potential contributions in advance by initiating a new discussion on GitHub Issues. This step fosters transparency and allows for preliminary feedback and coordination among the organisation members.
- **Branching Strategy for New Features/Issues.** For any new feature or issue, developers are required to create a branch directly from the relevant GitHub issue. This branch becomes the workspace for their changes. Such a practice ensures that the main branch remains stable and is only updated with thoroughly reviewed and approved changes.
- **Pull Request and Review Process.** Once the development work on a feature branch is complete, a pull request is created to merge the changes into the main branch. This pull request must be



assigned to a reviewer responsible for a thorough evaluation. The reviewer may accept the changes, request modifications or reject the proposal based on their assessment. Only after the reviewer's approval can the changes be merged into the main branch.

- **Maintaining a Clean Main Branch.** To preserve the integrity and continuity of the main branch, we strictly avoid introducing merge commits. All merges to the main branch must be fast-forwarded. As a best practice, we recommend squashing the commits on the feature branch before merging them into the main branch.

By adhering to these structured methodologies, we ensure that multiple team developers can contribute cohesively and effectively to the EDGELESS main repository, promoting a harmonious and productive development environment.

2.3 EDGELESS Integrated Framework

This section delineates the comprehensive approach to integrating the various components developed within the EDGELESS project. Given the unique nature of our project, with some developments occurring directly in the main EDGELESS repository and others in separate repositories, the integration process is multifaceted and tailored to each component. Table 1 outlines the specific integration methodologies for each element, ensuring a cohesive and efficient amalgamation of all project components. This strategy ensures that all components work harmoniously, supporting the project's overarching goals and objectives.



Table 1: Overview of EDGELESS system components and integration process

Component/Development	Description
Function Repository	The Function Repository is a component that will be used to store functions developed by the Function Developers (funcdev) and the workflows developed by the Application Developers (appdev), storing both the JSON definitions and compiled code. The development of this component is taking place in a separate repository and will be integrated into the EDGELESS system through a REST API (Function Repository API), which will be primarily connected to the edgeless_cli.
Monitoring system	The EDGELESS integration of the monitoring system embeds monitoring functionalities directly into its codebase, facilitating real-time data collection and analysis during runtime. This integration enables EDGELESS to interact with the monitoring system through API calls autonomously, ensuring proactive monitoring and swift responses to dynamic system conditions. Complementing the EDGELESS integration through our RESTful API, partners may gain access to the monitoring system, enabling them to monitor and analyse metrics related to EDGELESS.
Native execution run-time	Our approach to native execution run-time is designed to accommodate various forms of virtualization, tailored to suit a broad spectrum of computing platforms, including armv8, x86, and wasm architectures in the initial release. Its seamless integration will facilitate the operation of edgeless systems, allowing a wide range of function types to run across these specified architectures. This integral component will be seamlessly incorporated into the edgeless_node, ensuring smooth execution of use-case functions. Additionally, we will enhance the edgeless_cli with commands to locally compile code for diverse architectures. Moreover, a standardised Github action template will be provided to streamline and automate the compilation and release process. This will enable the Edgeless system to deploy workflows by fetching either the source code or the platform binaries directly from Github.
Authenticated registration	The authenticated registration will provide access control to the architecture's edge nodes. This process benefits from using a Hardware Secure Module (HSM), a dedicated hardware component that safeguards the cryptographic keys. During registration, the node proves its identity, which is backed and fortified by the HSM, mitigating the risk of tampering or impersonation, and only if it is valid and registered in the architecture access will be allowed. This authenticated registration process also allows for establishing a secure communication channel between the edge node and other elements of the architecture, adding additional security for data flows.
SGX execution	EDGELESS implementation of TEEs will ensure that functions can be executed from trusted hardware inside enclaves if needed. For this reason, small devices (like Intel NUC devices) will be exploited and take the role of edge nodes. These will employ Intel SGX as a trusted environment for running secure code. Function developers will not need to modify their existing code to implement trusted parts, only requirement should be to

	<p>specify which functions should run inside TEEs. Orchestrator, having that in mind, and being aware of specialised hardware with Intel SGX capabilities, will dispatch tasks to those devices dynamically. Additionally, utilising attestation principles will ensure that applications remain untampered and run only on genuine nodes, this information should also be provided to the Orchestrator which should use it to be aware of the system nodes.</p>
CUDA run-time	<p>CUDA run-time is an EDGELESS component that allows function developers to develop code that runs computations on GPU using CUDA. For seamless integration purposes, these functions need to follow EDGELESS API, which means that function developers should consider it before enforcing the required behaviours. One of the provided options is to develop actual implementations in Python by utilising libraries like numba, etc., due to the concrete establishment of well-known ML/AI frameworks there. Last but not least, from the function implementations, extra information (during workload creations) should be given to specify which parts must be routed by the orchestrator to GPU-powered hardware on runtime.</p>
Embedded Node	<p>The Edgeless Embedded Node is an implementation of a subset of the functionalities of the full Edgeless (worker) Node for resource-constrained microcontrollers—larger Linux-based embedded devices are supported by the full Edgeless Node. Its primary focus is the integration of hardware devices such as sensors and actuators, directly into the Edgeless System (without the DDA). We also aim to enable the execution of simple Edgeless Functions on these devices. The Embedded Node is developed as part of the main repository, and we try to share parts of the implementation with the full node. The limited nature of these microcontrollers still requires a custom implementation of many aspects of the embedded node.</p>
Computational container run-time	<p>The computational container run-time enables the execution of function instances within OCI containers. The lifecycle of the container is managed by the EDGELESS node, which also acts as a proxy for inbound/outbound events.</p>
Anomaly detection	<p>The anomaly detection component identifies when functions are saturated in the EDGELESS system. We will explore the best way to report these cases to the ε-ORC or SLA manager.</p>
ε-ORC delegated orchestration	<p>The delegated orchestration approach consists of the ε-ORC implementing only basic lifecycle management features for function instances, while the sophisticated orchestration logic is carried out by a companion component based on the set of active functions and their dependencies and annotations, the capabilities of the nodes in the orchestration domain, and the run-time metrics collected on the node/workflow/network utilisation.</p>
AI-based orchestration	<p>Working as a delegated orchestrator that makes short-term decisions based on a metric optimization process. It captures almost real-time information (currently Redis) about the current state of the EDGELESS workflows/functions on each and every node, capabilities and function</p>

	attributes specified in annotations, based on that monetization it returns updates of functions assignment.
SLA manager	The SLA Manager augments work-flows with SLA contracts to define runtime guarantees over intra or inter cluster processing. It mediates cross-cluster work-flow deployment. It pre-process the work-flow before forwarding it to the ϵ -CONs by for example translating a requirement into different lower-level annotations per cluster. It fetches cluster level aggregated or complete metrics from the ϵ -CON within each cluster. It merges the metrics to build the overall cross-cluster indicators. The SLA Manager might also fetch metrics from a cluster Monitoring System and raise notification when an anomaly is detected or might rely on a cluster Anomaly Detection component for detection. The SLA Manager raises notifications to the ϵ -CON to seek mitigation or alert the Application Developer. The SLA Manager may store and track the violations for auditing.
DDA	The Data Distribution Agent (DDA) is a component for the EDGELESS framework that enables: Interaction with various external data sources, sinks and services (such as sensors, actuators, industry protocols, enterprise IT systems and services), invocation of functions or workflows, delivering results via a data-centric approach and potentially supporting management of the state of distributed applications external to EDGELESS. The DDA achieves this by providing a common interface and abstractions for accessing heterogeneous external data sources. It is currently implemented as an edgeless resource that connects to a DDA sidecar.

2.4 Edgeless Open Source Repository

The main repository of the EDGELESS project is a cornerstone of our collaborative effort, reflecting a well-organised and logical structure essential for efficient project management and integration. The following table provides a detailed breakdown of the repository's structure, offering clear insights into the arrangement and purpose of its various components. This concise guide will help team members and stakeholders quickly understand and navigate the repository, ensuring smooth and effective collaboration across all aspects of the project.

Table 2: EDGELESS project main repository structure.

Directory	Description
documentation	Repository documentation.
edgeless_api	gRPC API definitions, both services and messages. This directory must be imported by other projects wishing to interact with

	EDGELESS components through its interfaces. The following interfaces are currently implemented: s01, s04, s06, s07.
edgeless_api_core	Work-in-progress development on minimal functions for embedded devices using CoAP.
edgeless_bal	Reference implementation of the ϵ -BAL, currently a mere skeleton. The concrete implementation will be done in the next project phase when inter-domain workflows will be supported.
edgeless_benchmark	Suite to benchmark an EDGELESS system in controlled and repeatable conditions using artificial workloads.
edgeless_cli	EDGELESS command-line interface. This is used currently to locally build function instances and to interact with the ϵ -CON via the s04 interface to create/terminate/list workflows.
edgeless_con	Reference implementation of the ϵ -CON. Currently, it does not support SLA/annotations but rather accepts any workflow and allocates all the function instances.
edgeless_dataplane	EDGELESS intra-domain dataplane, which is realised through the full-mesh interconnection of gRPC services implementing the s01 API.
edgeless_embedded	Work-in-progress implementation of special features for embedded devices.
edgeless_embedded_emu	Embedded device emulator.
edgeless_embedded_esp32	Support for some ESP32 microcontrollers.
edgeless_function	WebAssembly Rust bindings and function programming model.
edgeless_http	Utility structures and methods for HTTP bindings.
edgeless_inabox	Implements a minimal, yet complete, EDGELESS system consisting of an ϵ -CON, an ϵ -ORC, an ϵ -BAL and an edgeless node. This is intended to be used for development/validation purposes.
edgeless_node	EDGELESS node with WebAssembly run-time. Further run-times, possibly including OCI containers with Python/C++ support, are planned to be developed in the upcoming phase of the project.
edgeless_orc	Reference implementation of the ϵ -ORC. Currently, it ignores function annotations, and it implements two simple function instance allocation strategies: random and round-robin. Upscaling is not supported: all the functions are deployed as single instances.
edgeless_systemtests	Tests of EDGELESS components deployed in a system fashion, e.g., interacting through gRPC interfaces.

edgeless_telemetry	Work-in-progress component that provides telemetry data regarding the EDGELESS operation, also supporting Prometheus agents.
examples	Contains several examples showcasing the key features of the EDGELESS reference implementation.
model	Work-in-progress OCaml model of the EDGELESS system.
scripts	Collection of scripts.

2.5 CI/CD Processes

In this section, we delve into the crucial role of Continuous Integration (CI) and Continuous Delivery (CD) in our development pipeline. Recognising the significance of these processes, our approach is tailored to ensure efficiency, reliability, and speed in software delivery.

Continuous Integration in EDGELESS

As previously outlined, our project leverages GitHub repositories and the dynamic capabilities of GitHub Actions. GitHub Actions enable the execution of CI workflows triggered by specific actions in repository branches, such as pushes or commits. Within the EDGELESS project, we utilise these GitHub Action workflows to perform critical operations, and these include:

- **Code Validations.** To maintain code quality and consistency, validations will run against each commit. This will include syntax checks, code linting, and adherence to our coding standards.
- **Automated Testing.** Tests are automatically executed to ensure new code does not break existing functionalities. This will include unit tests, integration tests, and performance tests, providing a safety net that boosts developer confidence in making changes.
- **Build Process.** Upon successful completion of tests, the code is then built. This step will ensure that the code compiles correctly and any dependencies are appropriately managed.
- **Release Preparation.** When a build is successful, it can be tagged for release. This will include updating version numbers, generating changelogs, and packaging the code.

Continuous Delivery and the Role of Cluster Operators

In the realm of CD, the EDGELESS system adopts a strategic approach, particularly in deploying EDGELESS clusters. The responsibility of deploying these clusters primarily falls to Cluster Operators (infradev), a role mainly assumed by the Use Cases in pilot phases. To facilitate this, the EDGELESS system will provide tools designed to assist Cluster Operators in the deployment process. These tools will be designed to be intuitive and robust, ensuring deployments are smooth and error-free.

3 Use cases

In this section, the three UCs are described in detail, the main motivations and challenges from where UC requirements and the corresponding mapping with Edgeless features are presented.

3.1 Autonomous Smart City Surveillance (ASCS)

3.1.1 Description

This use case consists of a complete computer vision surveillance serverless application leveraging EDGELESS's orchestration and flexibility capabilities. The starting point for developing this platform will take inspiration from two Ubiwhere solutions already being developed i) the Intelligent Video Surveillance System (IVS), an autonomous computer vision framework capable of detecting and classifying objects based on real-time video feeds sourced from heterogeneous devices and transmission protocols and ii) the Ubi-OS is based on real-time sensor data processing, mainly correlating data inferring meaningful complex events.

In a common setting, one encounters numerous Smart Lamp Posts, each with CCTV Cameras installed, as exhibited in Figure 1. The pivotal function of these cameras is to relay live video streams to a dedicated local processing node, acting as an edge node. Notably, the edge nodes in this configuration are Nvidia Jetson Xavier, serving as the central processing units for the real-time video data emanating from the CCTV Cameras strategically positioned on the Smart Lamp Posts.



Real Scenario

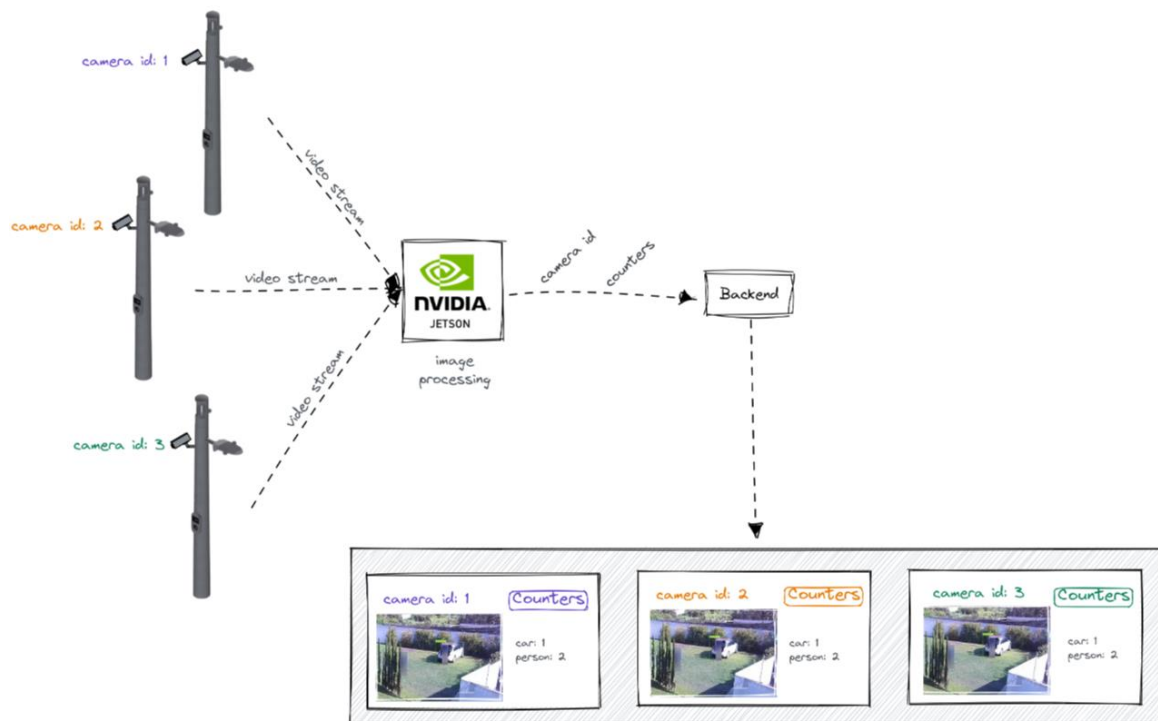


Figure 1: ASCS Real Scenario Overview

Providing a more detailed perspective on the edge node, various components operate locally, while specific data is transmitted to a central cloud-based platform. Machine learning algorithms detect objects such as vehicles or persons at the edge. The resulting counts for each identified object and person are then transmitted to the cloud. Additionally, a module is in place to monitor the power consumption of the edge node.

On the cloud side, a comprehensive dashboard, utilising Grafana in this instance, is accessible to users. This dashboard displays counters for each detected object and person and power consumption measurements for the edge node. The infrastructure in the cloud also encompasses a Redis Server and a Prometheus metrics generator to enhance the system's overall functionality.

To this end, data processing applications will be decomposed across the network (edge and cloud) mainly into four functions:

- I. pre-processing for movement detection: using computer vision, low-powered devices will recognise differences between consecutive video frames;
- II. on-edge processing of object classification: using computer vision (Deep Learning), this phase will detect and classify objects in a particular scene or set of frames triggered by movement detection;

- III. on-edge inference of object and context correlation: once the classification of objects takes place, this phase will infer the interaction amongst objects and provide possible actions that are occurring; the inference process also relies on AI, specifically Deep Learning, and a Complex Event Processing (CEP) engine that can provide a correlation of predefined rules and should be running on GPU-powered hardware;
- IV. on cloud generation of events: Immediately after a potentially hazardous action has been inferred, it will be reported to a central control application unit, which will record the event and enable human intervention to take the appropriate mitigation measures.

The 3 Smart Lampposts feature modular light poles equipped with various modules, including cameras, 5G small cells, EV charging stations, and more, highlighted in Figure 2, that can be easily installed. The CCTV Cameras are integrated into the lampposts, along with Raspberry Pis and NVIDIA Jetson Nano Developer Kit, boasting an AI performance of 472 GFLOPS, a 128-core NVIDIA Maxwell™ GPU, a Quad-core ARM A57 @ 1.43 GHz CPU, 4GB 64-bit LPDDR4 25.6GB/s memory, and microSD storage (card not included).

Furthermore, the lampposts incorporate the advanced NVIDIA Jetson Xavier module, delivering an impressive AI performance of 21 TOPS. This module features a 384-core NVIDIA Volta™ GPU with 48 Tensor Cores, a 6-core NVIDIA Carmel ARM®v8.2 64-bit CPU with 6MB L2 + 4MB L3, 16 GB 128-bit LPDDR4x memory with a speed of 59.7GB/s, and a 16 GB eMMC 5.1 storage solution, as shown in Figure 3.

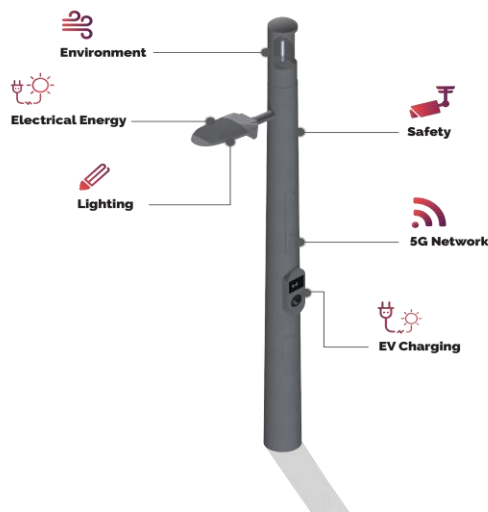


Figure 2: Smart Lamp Post example



Figure 3: NVIDIA Jetson Xavier and ANNKE 1080p 1920TVL Security CCTV Bullet Camera

In short, this UC presents a surveillance solution for municipalities to detect potentially hazardous situations that can put human lives at risk. Other target segments can include enterprises with distributed facilities wanting an integrated surveillance system with cloud and edge processing capabilities.

3.1.2 Motivation & Challenges

On a high level, Ubiwhere's use case always works the same. The CCTVs installed on the Smart lampposts gather the images, and each camera sends the live video stream to a local processing node (edge node), in this case, a Nvidia Jetson Xavier. On the edge server, further processing occurs (detection, faces and car plates anonymisation, i.e., blurred). And finally, data reaches the cloud for storage.

To validate EDGELESS, this use case aims to provide a solution for live stream data. The main goal is to use a nearby node whenever the first one is compromised, or there is any suspicion. In this case, the processing and storage will be sent to the nearby available node, and the first is temporarily shut down.

We can consider that the Smart City Analytics use case can be subdivided into two main sub-cases, depending on where the information is sent. So, with this in mind, we can consider the following sub-cases:

- Full edge sub-case;
- Hybrid edge-cloud sub-case.

This decision is dynamically/automatically made by an orchestrator, which takes into account metrics such as the safety of the data (e.g. safeguarding data throughout its entire life cycle to protect it from corruption, theft, or unauthorised access), the health of the system (i.e., if it detects something wrong with the jetson, for example) makes a decision of where the video stream is processed and where the result of the processing is stored, or an object is detected.

With this in mind, the orchestrator can entail two different key aspects of the edge computing paradigm: (i) latency sensitivity, in which the core objective can be to improve the quality of service, and (ii) latency criticality, where the network and edge proximity are essential for complex and critical use cases to be implemented. While latency-sensitive applications can result in degraded performance or user experience, the ones that rely on critical latency can result in major service failures. In this sense, these new applications and services are to be deployed on low-power and typically distributed constrained devices, and specific challenges apply that would not otherwise be met when compared to typically

centralised cloud computing solutions. Even more so, since Ubiwhere relies on these edge computing units to process large workloads of data and apply several AI/ML models at the edge, the integration of other EDGELESS nodes creates the need to rewrite the applications and foster more performance-driven solutions to allow for their faster implementation and overall edge scalability.

For the first scenario, the use case can provide a specific KPI on error rate: the percentage of data entries containing errors. Finally, the system's health can be checked with a monitoring component such as Grafana that can depict the deployed components' overall status and usage. This solution provides an overview of GPU usage, an essential metric for Computer Vision development.

The ASCS UC will require an intricate interplay between edge nodes, their associated applications, and the seamless integration with cloud resources. Potential disruptions in the operational status of edge nodes may occur due to various factors such as software upgrades, hardware malfunctions, or transient network connectivity issues. It is intended that Edgeless handles these events that can cause service degradation, and dynamically adjust application parameters, including the reduction of frame rates or the implementation of quantisation for machine learning parameters. These adaptive measures will enable the management of the workloads, adapting to scenarios with constrained computational resources.

Moreover, the ASCS UC intends to leverage inter-cluster reorganisation facilitated by the ϵ -controller. This involves the judicious transfer of specific function executions to another peer cluster, leveraging pre-established business agreements to optimise workload distribution across clusters to assure a good level of performance. The ϵ -controller orchestrates the relocation of specific function executions to the cloud, capitalising on the additional resources available. This versatile utilisation of both edge and cloud resources represents a very important functionality for this UC, ensuring robust resilience in the face of evolving operational conditions.

To summarise, the ASCS UC will look for Edgeless to answer the following requirements:

- **Availability**, by shifting or moving workloads from edge node to edge node and/or to the cloud, it is very important that the services and applications are kept alive and running. Data should be available in a distributed manner for redundancy, and in case hazardous events occur that cause edge nodes outages, Edgeless should be able to detect anomalies, through the anomaly detection and monitoring features, while also providing data and function forwarding mechanisms that assure no data is lost and the impact on running services and applications is mitigated.
- **Scalability**, since at a city level we can expect to have hundreds if not thousands of edge nodes and video streams, scalability is a must have for this UC. With resources being more scarce on the edge, Edgeless should dynamically optimise the use of resources, to maintain good performance levels for the running workloads. This matches with the resource orchestration and function run-time scheduling features of Edgeless
- **Privacy and Security** through Edgeless trusted execution environment, sensitive data should be protected and secure, essential in maintaining people's trust in the surveillance solution. Furthermore, edge nodes are much more exposed when compared with data centres. Edgeless should provide mechanisms to secure node identification, avoiding spoofing attacks. It should also

detect if an edge node is compromised, while taking the necessary actions to avoid data leaks and isolating the threat from other edge nodes.

- **Hardware heterogeneity**, by supporting multiple types of hardware that range from a Raspberry Pi or an NVidia Jetson, placed at the edge, and more powerful computing platforms like cloud servers. Applications and workloads should move seamlessly from one computing platform to, benefiting from the serverless computing capabilities of Edgeless.

3.1.3 Edgeless Workflow

The use case will be divided into two sub-cases: full edge and hybrid. Following the full edge sub-case, the ASCS UC relies on processing and storing information at the edge. This intends to remove the need to send and store information in a cloud server that can be more easily compromised than several edge nodes deployed on an edge network.

With this in mind, the Use Case will provide Edge security to prioritise the most critical security fundamentals, including authentication and authorisation mechanisms, for data stored locally and in transit between the network core and edge computing devices.

- In this sub-case, the idea is that the processing and the storage happen all in the edge node. So, the CCTV camera gathers the video stream and sends it to the edge node, a Jetson Xavier. It will be there that the frames will be processed to detect vehicles and other objects. Then, it will only store necessary metrics that are indispensable for city management regarding damage to infrastructure and public buildings.
- The hybrid sub-case happens when the edge is compromised or the detected object or scene is not mandatory to move the processing to the cloud. Also, this sub-case can be used when processing the video stream on the edge is unsafe. In this case, the video stream is sent to the Computer Vision algorithms deployed in the cloud, and then the results of the algorithms are stored in the cloud.

The transition from the cloud to deploying resources and services closer to the end-users aims to bring computing resources and services closer to the edge devices, either by deploying these resources in regional data centres or directly at the edge. Consequently, by design, the edge infrastructure is extremely distributed to cover all the users over the coverage area.

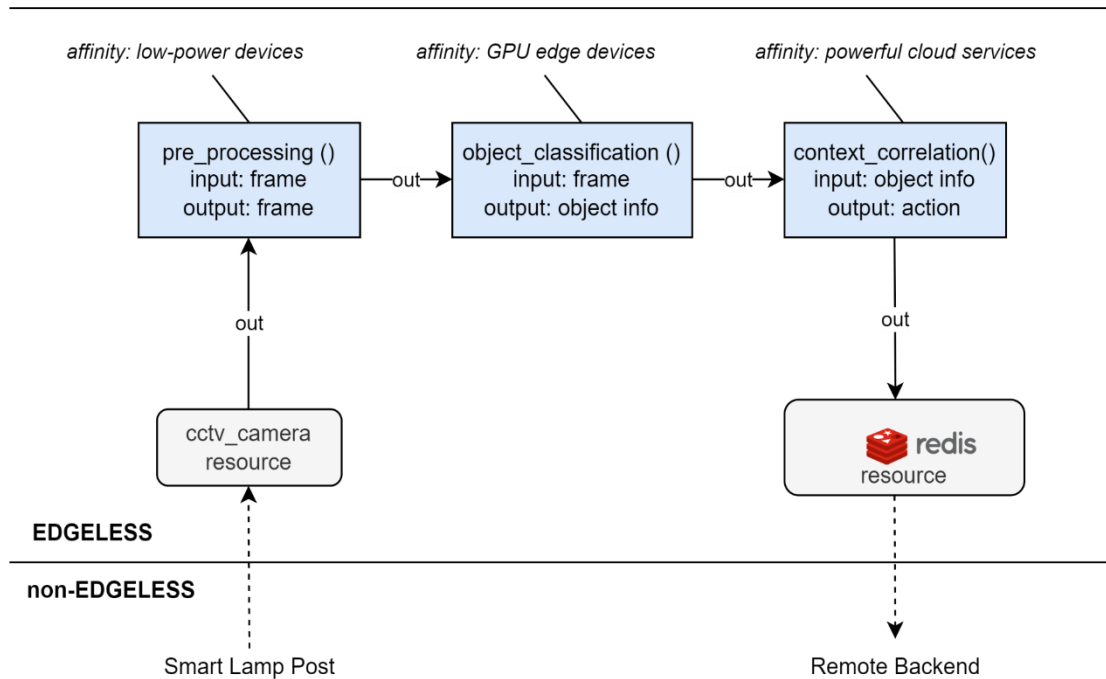


Figure 4: Autonomous Smart City Surveillance Use Case architecture

3.1.4 Open points & next steps

With the requirements identified, in the short term the ASCS UC aims at integrating ML models within the Edgeless workflow, the redis implementation and workflow management. The macro steps of the roadmap are the following:

- Full-edge implementation at a laboratory setup
- Test and validate Edgeless at the laboratory setup
- Physical installations in the SmartLampPosts
- Deployment at the UC location
- Test and validate Edgeless at UC location

Open points

Our primary interest revolves around delving into the intricate process of categorising various elements within the purview of EDGELESS. This includes meticulously examining edge nodes, devices, and applications and identifying the pertinent attributes that demand consideration. These attributes include metrics such as CPU, RAM, and Storage alongside critical performance metrics such as the required latency for each application.

This categorisation process seamlessly intertwines with the earlier point, underscoring the imperative of harmonising edge nodes, device onboarding, and application management. It necessitates a thorough comprehension of each element's distinct characteristics and prerequisites, ensuring that EDGELESS is equipped with the requisite information to effectively coordinate and optimise the functionalities of edge nodes, devices, and applications. By scrutinising specific factors like CPU, RAM, Storage, and latency considerations, we aim to refine the overall categorisation process, empowering the EDGELESS system to navigate its ecosystem with precision and efficiency.

Testing and validation

Based on the identified requirements, the ASCS UC will attempt to perform tests that can validate the Edgeless features that answer these requirements. At the current stage it is difficult to detail a testing plan, however it is intended to include actions such as:

- Intentionally shutdown an edge node, simulating a node outage, and evaluate how Edgeless responds. Metrics such as downtime and recovery time can be useful in assessing Edgeless performance.
- Overloading an edge node with heavy workloads, assessing how Edgeless transfers some of the functions to other edge nodes that are less congested. Metrics such as the number of loss frames can help in assessing the mitigation provided by Edgeless in these scenarios.
- Use simulated data such as replicated video streams to assess Edgeless scalability functionalities
- Attempt to register a rogue edge node and assess if Edgeless recognizes the threat and responds accordingly, protecting data and integrity of the remaining edge nodes.
- Run Edgeless in various resource constrained devices, such as Raspberry Pis, NVidia Jetsons.

3.2 Internet of Robotic Things (IoRT)

3.2.1 Description

The main goal of this use case is to validate the EDGELESS framework and systems within an IoRT pilot, emphasising on the integration of autonomous robotics and IoT-connected sensors in agile manufacturing scenarios. This involves executing AI-scheduled and low-latency tasks within the production workflow.

The use case scenario environment involves a combination of stationary and mobile robots equipped with IoT-connected sensors, such as cameras, pressure sensors and laser scanners, operating collaboratively in the production environment. The EDGELESS framework orchestrates the operation of robotics equipment, dynamically assigning tasks based on real-time data and production priorities. By taking advantage of AI-driven algorithms, the framework optimises task scheduling, resource allocation, and workflow coordination to ensure efficient utilisation of manufacturing resources.



In addition to the robots, the hardware shown in the table will be used for planning and executing tasks within this IoRT application. Below is Table 3 outlining the key hardware components planned for utilisation:

Table 3: Key hardware sensors planned for the Internet of Robotic Things Application.

Hardware	Attribute	Value
NVIDIA Jetson Nano Developer Kit	AI Perf	472 GFLOPS
	GPU	128-core NVIDIA Maxwell™
	CPU	Quad-core ARM A57 @ 1.43 GHz
	Memory	4GB 64-bit LPDDR4 25.6GB/s
	Storage	microSD (Card not included)
ORBEC Astra Pro 3D Camera	Range	0.6m - 8m
	FOV	60°H x 49.5°V x 73°D
	Image resolution	1920 x 1080 @30fps (rgb), 640 x 480 @30fps (depth)
	Precision	+/- 1 – 3mm @1 m
	SDK	Astra SDK or OpenNI 2 or 3rd Party SDK
PLC Siemens 6ES7214-1HG40-0XB0	CPU	1214C
	Protocol	Profinet
	Program memory	100KB
Security laser SICK S3000 Standard	Reach	4m
	Scanning angle	190°
	Configurable resolution	30mm, 40mm, 50mm, 70mm, 150mm

	Angle resolution	0.5°, 0.25°
	Response time	60ms

Critical tasks, including object detection, classification, and workspace segmentation, will be offloaded to edge nodes for execution with minimal latency. This enables timely decision-making and seamless integration of data-driven insights into the manufacturing process.

The IoRT pilot aims to develop decentralised manufacturing systems embedded with adaptable cognitive capabilities, seeking to validate the potential of the EDGELESS framework in manufacturing environments, operating in real-time scenarios. Furthermore, the deployment of a robotic application in agile production will validate the feasibility and suitability of the results concerning the project objectives. Various robotic equipment and multimodal sensors will be combined, and real-time task scheduling will be managed by a system that operates the equipment dynamically, capitalising on the advantages offered by the EDGELESS architecture.

3.2.2 Motivation & Challenges

The Internet of Robotic Things Application will be run in the EURECAT laboratory, providing a representative environment similar to a realistic manufacturing scenario. Specifically, it will be demonstrated in an electric car battery disassembly scenario, where a stationary robot will be mounted on and communicate with a linear axis to extend its operating reach range. Equipped with cameras and sensors, the robot performs various disassembly tasks, such as screwdriver tasks and removing covers with a real-time dynamic path planning.

Rather than relying on predefined trajectories, the robot dynamically plans and executes its trajectories in real time to reach the target while avoiding collisions and re-planning if the system detects operators in the trajectory. As the battery disassembly process is very complex, there will be operators to assist in specific tasks. This is the reason why the system will have to know the tasks to be performed, identify the current task that the operator is performing and intervene and interfere depending on the position of the operator in the station and the task being executed.

Therefore, the robotic system's responsibilities will include object detection and classification, human detection and tracking, task identification, workplace segmentation and manufacturing shopfloor mapping. Therefore, the Edgeless system will enable the successful implementation of this use case. Below are listed the specific identified requirements for the IoRT application and how the unique capabilities of EDGELESS can be adapted to meet these needs:

- **Real-time trajectory planning and execution.** In this application robots need to plan and execute their movements in real-time while adapting to environment changes. EDGELESS offers low-latency capabilities using edge nodes close to the robots for fast decision making and providing high performance for planning movements.

- **Persistent functions.** The use case application requires functions that can retain past data to analyse sensor data over time with the aim of identifying activities and anomalies. EDGELESS supports these persistent functions, enabling the system to manage and analyse continuously data streams.
- **Scalability.** As the IoRT use case grows, it must handle increasing workloads. EDGELESS is able to scale up easily adjusting resources to support different workloads and integrating new functionalities into the system.
- **Machine learning models execution.** The application needs to run machine learning models on edge nodes for quick decision making. EDGELESS supports this by running machine learning models efficiently deploying and executing algorithms on edge devices.

3.2.3 Edgeless Workflow

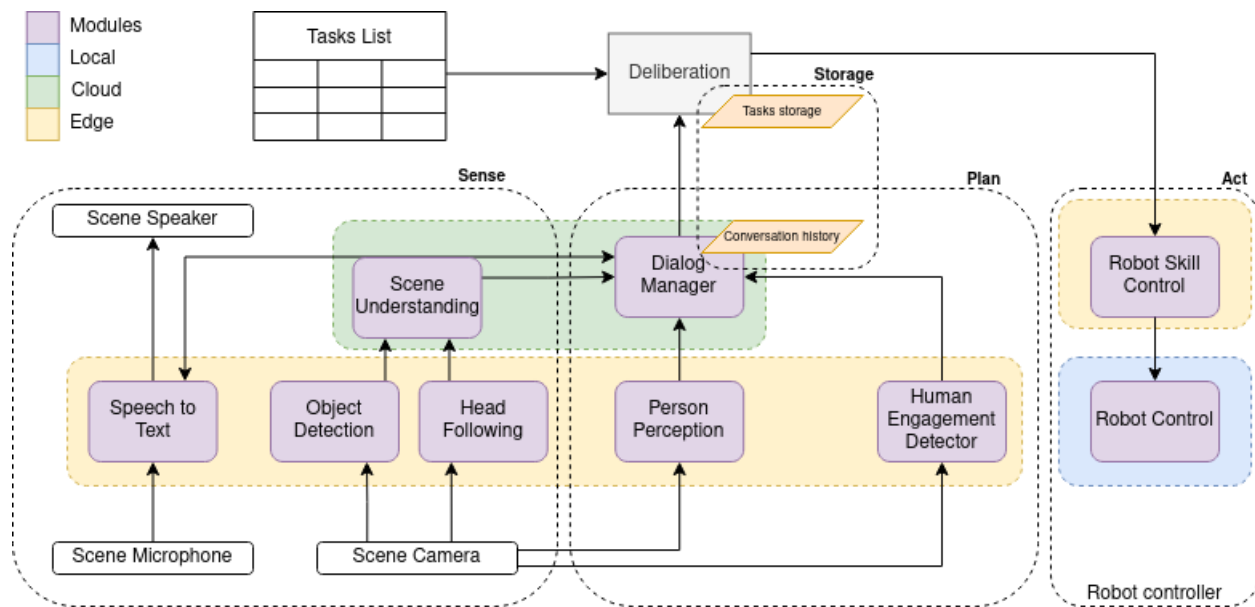


Figure 5: IoRT UC Architecture diagram integrated with EDGELESS workflows

Figure 5 shows how the use case will take advantage of the EDGELESS system. For the IoRT application, various elements must be coordinated to achieve an efficient execution.

In the use case scenario there are several physical hardware elements, such as robots, cameras, sensors, microphones, and speakers, among others, that are waiting for an input to be activated. Once this input is received, several use case modules will be executed on the edge. These modules, such as speech to text, object detection, person perception, among others, will send their output to other physical elements or additional modules located in the cloud, such as scene understanding and dialog manager.

The robot behaviour will be designed to perform a series of tasks to achieve the correct use case execution. Taking into account the different tasks that need to be executed and the computed outputs of each module, the robot system will apply a deliberation layer to decide how to act at each moment. After

this, the execution of the robot control module will occur internally to the robot, taking care of planning and executing the robot's trajectories.

To better understand the different elements of the use case and their functions, a list and a description of the principal ones is listed below.

- Cameras:
 - Capture video in real-time.
 - Send the captured data to the local processing node (edge node).
 - Process and extract information from the cameras.
 - Identify the objects in the workspace.
 - Obtain the operators' and robots' positions and the tasks in execution.
- Robots:
 - Receive information from the local processing node (edge node).
 - Depending on the information they receive, they make different decisions.
 - With the information received, they check for a clear trajectory.
 - If there are obstacles in the trajectory:
 - They re-plan the trajectory in real-time.
 - If this is impossible, they stop and wait.
 - If there are no obstacles:
 - If any operators are detected nearby, they go to execute another task and recheck for obstacles.
 - If no operators are detected, the robots execute the planned task.
 - The trajectories are planned and executed in real-time and calculated in the robot controller (locally).
- Lasers and other components feed the system with additional information and thus contribute to the overall status of the station.
- Data from the environment is processed in real time. Relevant data is collected and sent to the cloud server.

Obstacles are defined as events triggered by proximity lasers, some information that makes the robot stop, such as an operator detected in the area, or loss of information, among others. In this case, if the robot can replan and execute the trajectory without endangering the system components, it continues to execute the task until it completes it. In any other case where it cannot replan, the robot stops and returns to process the information from the edge node.

To better understand the different processing elements of the use case, a list and a description of the principal ones is listed below.

- Robot and devices controllers:
 - Internal or external controllers that manage information and make robot actions and decisions.
- Edge nodes:

- Local processing nodes that process video and other information and send it to the robot agents.
- Cloud:
 - System that stores the data received from the different agents in the scenario.
 - It displays different statistics and information.

Throughout the execution of the robot, information from the environment is read and published to the cloud and the edge node.

This use case takes advantage of the EDGELESS system capabilities by efficiently integrating the execution of functions locally, at the edge and in the cloud. Executing the robot function modules on the edge a faster response can be achieved, as well as a latency reduction. By using the cloud for more demanding tasks, resources are leveraged, ensuring additional processing power when needed.

3.2.4 Open Points and Next Steps

As we progress in the IoRT use case, the next steps focus on developing the different robotic modules within the EDGELESS workflow. The immediate next steps involve deploying an EDGELESS system at the laboratory and implementing edge functionalities. Then, we will proceed to test and validate the EDGELESS system at the laboratory setting before testing and validating the system in the actual use case scenario.

- Deploy an edge system at the laboratory.
- Edge implementation at the laboratory.
- Test and validate the EDGELESS system at the laboratory.
- Deploy the EDGELESS system in the use case scenario.

Open points

- The design of EDGELESS robotic functions must align with EDGELESS protocols and standards to ensure seamless communication between robotic components and the EDGELESS infrastructure.
- Preprocessing sensor data is critical in robotics to make decisions and execute tasks accurately. EDGELESS must integrate state functions to effectively manage temporal windows and enable accurate pattern recognition and anomaly detection in real-time robotic operations.
- Optimising the EDGELESS deployment for robotic applications involves the strategic placement of hardware components and the design of the system architecture to maximise performance and integration with robotic systems. Decisions regarding the deployment approach will require careful consideration.

Testing and validation

For testing and validation of the use case application, a simulated environment will be created in the EURECAT laboratory that simulates a realistic manufacturing scenario. The robot, equipped with cameras and sensors, will perform manufacturing tasks in real time, planning and executing trajectories dynamically.

The main test scenarios are the intentional shutdown of peripheral nodes to simulate outages, the overloading of peripheral nodes to evaluate workload distribution, and the use of simulated data streams to evaluate scalability.

The whole application operation will validate EDGELESS functionalities. This testing ensures that the EDGELESS system fulfils the requirements of IoRT applications and performs effectively in real manufacturing scenarios.

3.3 HealthCare Assistance (HCA)

3.3.1 Description

This UC aims to design and develop a personal assistant with activity identification and anomaly detection capabilities. To achieve this goal, a system will be developed that will use IoT devices (several sensors and a gateway) located in the homes of the elderly people for whom the system is intended.

Through the data collected by multiple sensors of different types (motion sensors, pressure sensors, etc.) located in the rooms of the house, the system will be able to recognise the activities performed by people and detect anomalies that may occur.

Once the activities have been identified, this data will be sent to a component responsible for managing this information and allowing access to caregivers and family members. If an anomaly is detected, this information will be sent to a component that will notify caregivers and family members. Furthermore, this data can be consulted at any time. To access this data, the system will provide a front-end application accessible from any device.

The physical infrastructure will be located in the persons' homes (seniors, pre/post-surgery or chronic disease patients).

The system will be designed to incorporate a variety of sensors strategically placed throughout different rooms in a home. Each type of sensor is tailored to monitor specific aspects of the living environment, ensuring comprehensive data collection.

Figure 6 shows an example of how the sensors will be located in the house.

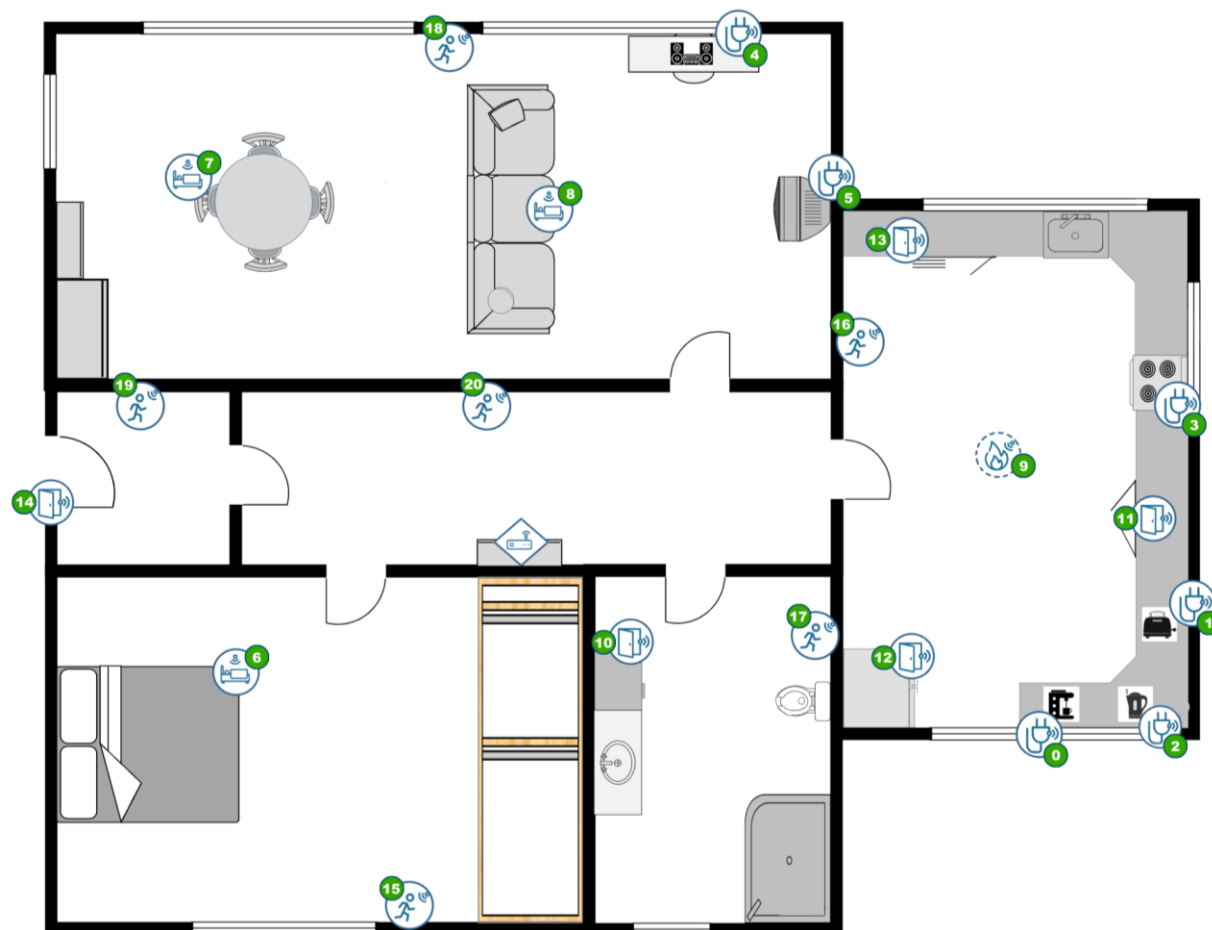


Figure 6: Person's home with sensors depicted

The system features a range of sensors connected to a gateway. These sensors, diverse in their functions, will gather data, while the gateway will act as the point for communication with the EDGELESS system. Furthermore, the system will take advantage of the 5G/MEC testbed provided by Telefónica.

Gateway:






Raspberry Pi 4/5.

Sensor Devices:

There are sensor types that generate data only when something changes/is detected and other sensor types that generate data every minute.

Table 4: Overview of the sensor types

Device	Details
--------	---------

	Mattress Device	<p><u>Type:</u> Pressure sensor.</p> <p><u>Event Trigger:</u> The sensor is activated upon each change of state, detecting when pressure is applied or removed.</p> <p><u>Item Attachment:</u> Can be attached to beds, couches, chairs, and similar furniture items.</p>
	Motion Device	<p><u>Type:</u> Motion sensor.</p> <p><u>Event Trigger:</u> Sends an alert with each detection of movement or cessation of movement within a room.</p> <p><u>Item Attachment:</u> Installed on room walls or ceilings for optimal coverage.</p>
	Smart Plug Device	<p><u>Type:</u> Electric power sensor.</p> <p><u>Event Trigger:</u> Reports electric consumption data every minute.</p> <p><u>Item Attachment:</u> Connected to electrical appliances such as TVs, coffee makers, music systems, and lamps.</p>
	Window-Door Device	<p><u>Type:</u> Contact sensor.</p> <p><u>Event Trigger:</u> Activates when the contact between two device components is broken or re-established.</p> <p><u>Item Attachment:</u> Suitable for doors, windows, boxes, and drawers to monitor access and closure.</p>
	Smoke Device	<p><u>Type:</u> Smoke presence sensor.</p> <p><u>Event Trigger:</u> Triggers only when smoke is detected to alert for potential fire hazards.</p> <p><u>Item Attachment:</u> Fixed to ceilings, the device ensures timely smoke detection.</p>

For the HealthCare Assistant (HCA), multiple sensors would be connected to a Gateway in the senior person's home, then the Gateway will be connected to the Internet. For the Pilot, we are going to simulate this scenario instead of having a real home with the sensors.

The senior person's simulator is going to be generating data directly in the gateway, simulating the scenario shown in Figure 6.

Testbed

This UC will use the testbed provided by Telefónica, highlighted in Figure 7, to realise a stand-alone cluster.

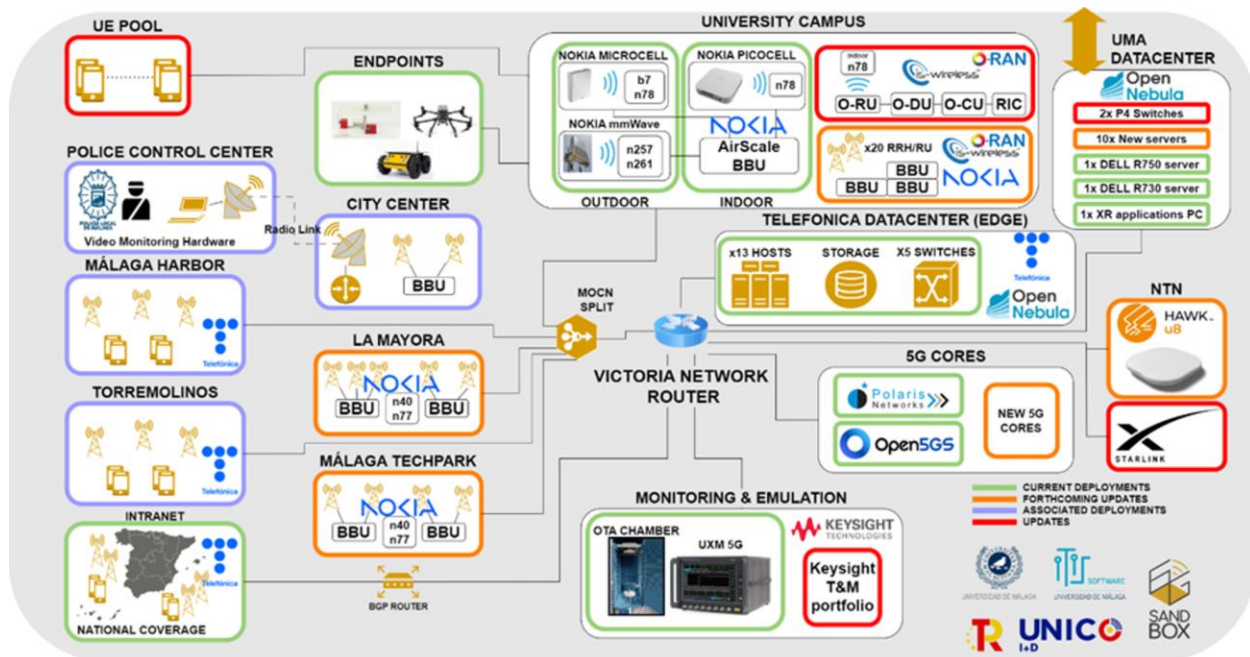


Figure 7: Malaga Platform (Victoria Network)

Telefonica has two micro data centres based on hyper converged architecture (HCI), detailed in Figure 8. Hyper Converged infrastructure (HCI) combines common data centre hardware that uses locally attached storage resources with smart software (SDS) to create flexible building blocks that replace traditional infrastructure of servers, storage networks, and separate array storage.

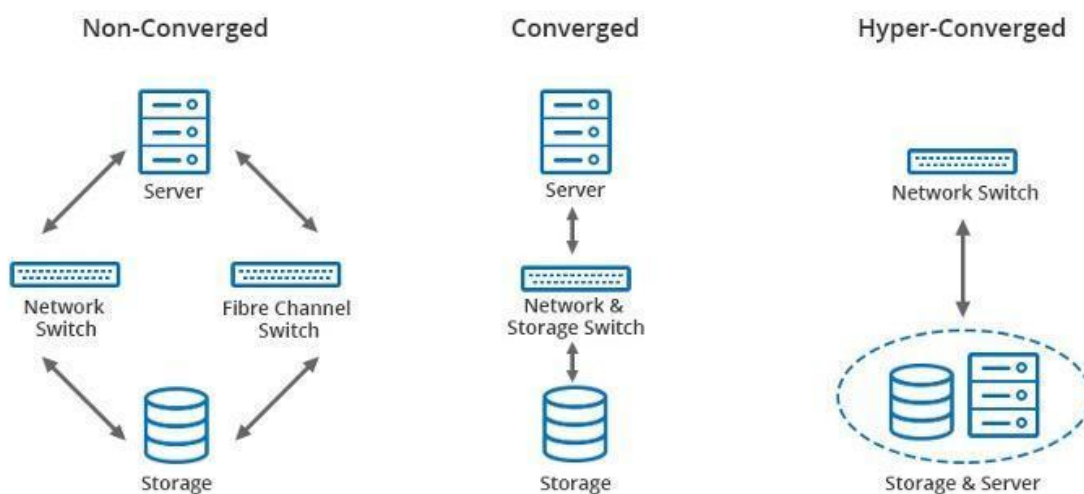


Figure 8: Data Center Architectures

The solution chosen to manage the private cloud management layer has been the OpenNebula (ONE) project. It is an open-source solution for creating hybrid clouds that has very interesting advantages. The OpenNebula solution allows us to:

- Manage different computing environments.
- VM
- Containers
- Kubernetes clusters
- Hybrid hardware
- Private nodes
- Public nodes
- Edge Nodes
- Unify the management of all your resources from a central point
- Federation
- Simplicity

The fact that it is an open source project supported by other European projects, with which Telefonica has been collaborating for many years, and that offers all the necessary characteristics required by the services and platforms that are going to be deployed in the testbed, has been the main decision to opt for this solution.

The microdata centre is split into two different locations: UMA and Peñuelas. The reason for splitting the data centre into two was to offer different capabilities in each of them and facilitate certain types of experiments and integrations.

UMA microdata Center:

It is in the UMA facilities.

- 11 physical hosts
- 668 physical cores
- 2.6 TB RAM memory
- 13 TB Shared Storage (CEPH)
- 1 x GPU Nvidia A100
- 2 x GPU Nvidia A2

Peñuelas microdata Center:

It is in the Telefonica Central Office called Peñuelas

- 8 physical hosts
- 448 physical cores
- 2.3 TB RAM
- 26 TB Storage (CEPH)
- 10 x GPU Nvidia RXT 2070 Super

Both data centres are interconnected through fibre optics and the NTN satellite connection and are prepared to deploy any type of service or platform using the same techniques and tools as if we did it in a public cloud.

3.3.2 Motivation & Challenges

The key motivation for this system lies in its ability to run detection algorithms on edge devices. This approach offers several advantages:

Firstly, it ensures low-latency processing and response since it is not necessary to send the sensor data to a remote server. Therefore, both activities and anomalies can be detected with low latency, allowing caregivers and family members to have the information and act in the shortest time possible if necessary.



Secondly, it enhances privacy, as sensors' data stays within the home. This project aims to leverage these benefits to provide a seamless and secure experience for monitoring daily activities and detecting anomalies in a home environment.

Additionally, incorporating a 5G/MEC testbed into the system is a strategic decision. The high-speed and low-latency characteristics of the testbed will enable a more efficient and fluid execution of the workflows. This is particularly important for the accurate and timely detection of anomalies.

For this Use Case, we have identified several key requirements essential for the system's proper functioning. While some of these requirements might be met with existing solutions, the unique combination of all these elements is not currently addressed by any single solution. Therefore, the EDGELESS system is the optimal system to implement this use case effectively. Below, we detail the specific requirements that have been identified, highlighting how the EDGELESS system's unique capabilities are ideally suited to meet these complex and integrated needs.

- Low-latency processing and response. Both activities and anomalies should be detected with low-latency, allowing caregivers and family members to have the information and act in the shortest time possible if necessary. The EDGELESS system will be able to provide low-latency capabilities by processing edge nodes that are close to the senior persons' homes, and providing an environment with high performance execution for the Rust programming language.
- Privacy and security. The data generated by sensors constitutes private information belonging to individuals, and as such, it is imperative that these data remain strictly confidential to protect personal privacy. The EDGELESS system will be able to fulfil this requirement by several ways, such as allowing us to execute functions that anonymize the data before the data leaves the home.
- Functions with state. Identifying activities and anomalies requires not only evaluating the data from the latest sensor detection but also considering all data from every sensor in the house within a specific time window, necessitating the maintenance of a state to encompass this comprehensive analysis. The EDGELESS system provides stateful functions that can be used to fulfil this requirement.
- Scalability. Due to this UC is intended to be used in the homes of senior people, the system could be installed in multiple homes. The system should be able to support the computational loads when new homes are integrated in the system. The EDGELESS system will be able to scale when needed.
- Execution of ML models through containers on edge nodes, including Cuda support. These containers should be able to receive an input and provide an output. The EDGELESS system will provide support for Docker containers and several languages, and we'll be able to execute a function in nodes that support Cuda by defining this in through annotations.
- Connection with external REST APIs through HTTP requests, to be able to connect the system to an existing platform in the Cloud (Senior Care). The EDGELESS system provides "Resources" that can be used to fulfil this requirement.



As we delve deeper into the intricacies of this Use Case, it's crucial to acknowledge and address the challenges that lie ahead:

1. Identify and cluster activities of daily living (eating, washing and grooming, toileting, sleeping), instrumental activities of daily living (meal preparation, housekeeping, laundry, medication use), ambulatory activities (walking inside-outside and up-down, stationary activities like sitting/standing/lying, transitional activities (sit-to-stand, stand-to-sit, stand-to-walk).
2. Detect anomalies representing a potential issue on the senior autonomy, well-being and safety.

3.3.3 EDGELESS Workflow

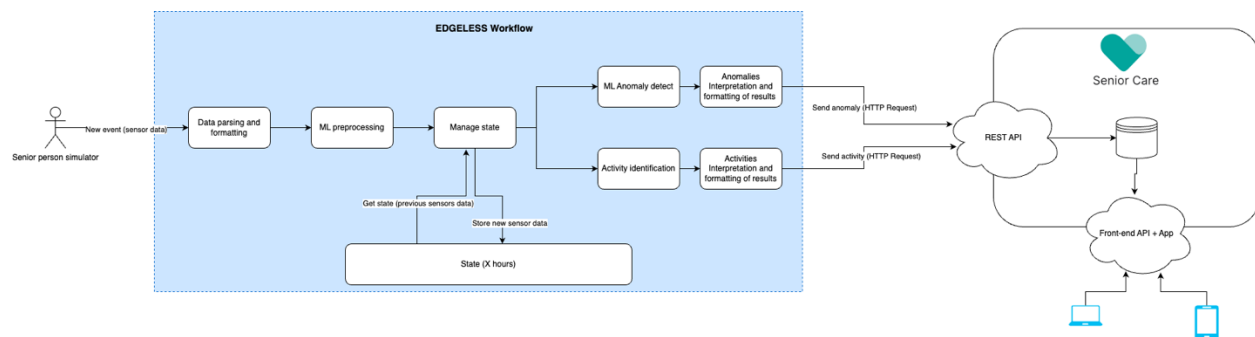


Figure 9: UC Architecture diagram integrated with EDGELESS workflows

Figure 9 shows how the UC is going to take advantage of the EDGELESS system, triggering the execution of an EDGELESS workflow every time a sensor generates data (for the pilot, this will be done with a simulator). The workflow described in Figure 9 represents a first version that may change over time.

Workflow invocation rates depend on the senior person's activities and are not always the same. There will be cases where there will be a higher rate of workflow invocations because of many sensor detections (moving around the house or cooking), and there will be cases where there will be a lower rate of workflow invocations (sleeping).

The sensor's data is ephemeral and must be kept only in the function's state for a time. The activities and anomalies detected will be persistent and only accessible by caretakers and family members.

Every time the sensor simulator generates data, it triggers the invocation of a workflow with the following functions:

Data parsing and formatting: Standardises incoming sensor data to ensure consistency across different devices and formats. This function will be developed with Rust.

ML preprocessing: data normalisation, feature creation, etc. This function will be developed with Rust.

State Management: Adds new sensor data to the state, retrieves previous detection data, and purges data older than the defined temporal window (to be determined in hours). It is also responsible for cleaning unnecessary data. This function will be developed with Rust.

Activity identification: A Pre-trained machine learning (ML) model evaluates the data to recognise the daily activities of the residents, providing insights into their routines. This detector will use Node.js by using a Docker container. This Activity detector will consist of a Tensorflow.js pre-trained model that can detect any of a set of possible activities (classification algorithm).

Anomaly Detection: A machine learning (ML) model processes sensor data to identify patterns that may indicate emergencies or health risks. This detector will use Python by using a Docker container. For the development of this Anomaly detector, several technologies and algorithms will be tested to ensure optimal performance and accuracy, such as VAE (Variational Autoencoders), ensemble algorithms (Isolation Forest or Random Forest) and statistical methods (e.g. Gaussian Mixture Models).

Anomalies Data Interpretation and Formatting: Processes the output from anomaly detection, formatting it suitably for REST API consumption by Senior Care App. This function will be developed with Rust.

Activities Data Interpretation and Formatting: Processes the output from activity identification, formatting it suitably for REST API consumption by Senior Care App. This function will be developed with Rust.

Data Transmission: Finalised data will be transmitted to Senior Care via HTTP requests for further use. This function will be developed with Rust.

Data Management and Monitoring in Senior Care (Cloud): Upon reception, the Senior Care will store the activities and anomalies detected in a database, where it is stored and managed for accessibility. It is important to note that all data used for the pilot will be sourced from a simulator, ensuring no real personal data is involved.

Caregivers will be able to monitor the collected information through a dedicated front-end application, enabling them to respond promptly to emergencies and track daily activities.

The architecture of this UC takes full advantage of the capabilities of EDGELESS. In summary, every time a sensor generates data, it will trigger the execution of an EDGELESS workflow, which will be responsible for recognising activities and detecting anomalies. Once the activities and anomalies are detected, this data will be sent to the Senior Care system through a REST API. Figure 10 shows this interconnection summary.

To connect what belongs to the EDGELESS system with what's outside of EDGELESS, we will use:

- **Before the EDGELESS workflow.** From the sensor's data generated by the Senior person's simulator, an EDGELESS workflow will be triggered. We'll use the *EDGELESS http_ingress resource* to implement that connection.
- **After the EDGELESS workflow.** Once activities and anomalies are detected in an EDGELESS workflow, that data will be sent to the Senior Care system. We'll use the *EDGELESS http_egress resource* to implement that connection.

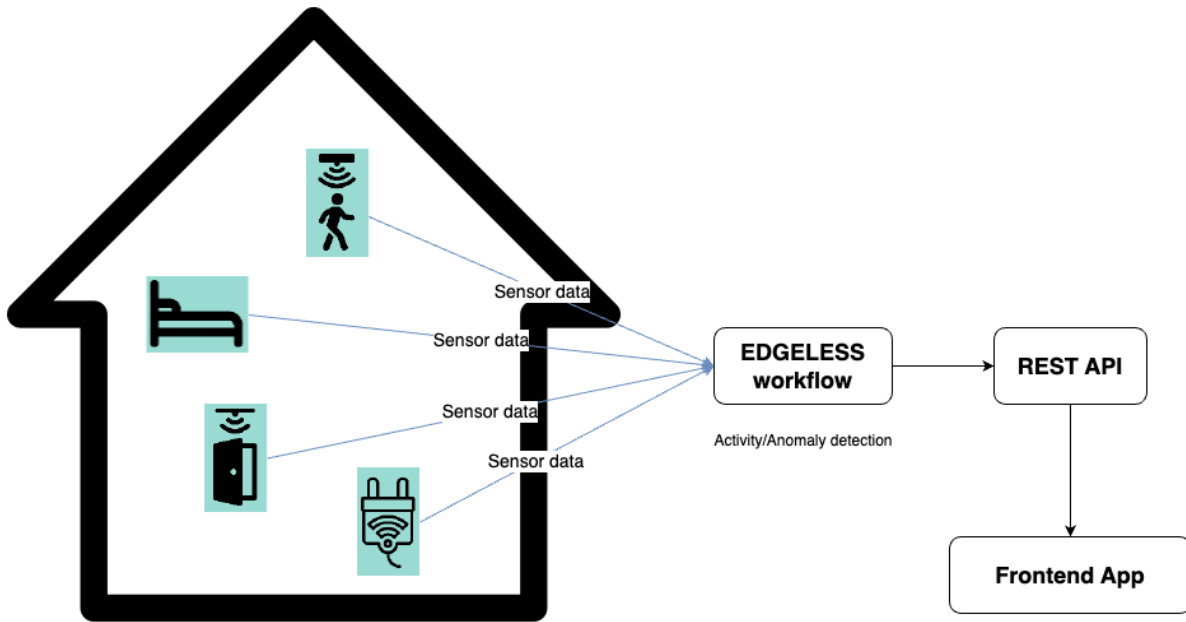


Figure 10: UC Interconnection diagram with EDGELESS

3.3.4 Open points and Next Steps

As we advance in the Use Case, the immediate next steps focus on the development of the first EDGELESS functions. Once these functions are completed and working within EDGELESS, we'll focus on the ML models, followed by the integration with the Senior Care App and utilising the testbed for thorough evaluation. This is the roadmap for the HCA Use Case development:

- Develop a senior person simulator.
- Define feature engineering for the simulator's data.
- Develop first Rust functions for the EDGELESS workflow.
- Develop and train a ML model for activity detection.
- Develop and train a ML model for anomaly detection.
- Integrate ML models within the EDGELESS workflow.
- Integration with the testbed provided by Telefónica.

- Integration with the current Senior Care app.
- Final tests and validations.

Open Points

1. The ML preprocessing is a key point in this Use Case and something we still need to address. This is important because it involves transforming sensor data into a format suitable for ML algorithms. Our main concern is the temporal analysis because the system needs to understand activities over time, not in isolation. Preprocessing will create a 'temporal window', which will require using the EDGELESS stateful functions, allowing algorithms to detect patterns and anomalies based on sequences of sensor readings.
2. Define the best approach to deploy EDGELESS in this UC, considering that there will be hardware (gateways) located at the senior persons' homes and we will use the testbed provided by Telefónica. The main reason for this open point comes from the need to find the best approach for this deployment, in which we could either deploy a single EDGELESS cluster with several orchestration domains, or deploy several EDGELESS clusters interconnected.

Testing and Validation

For the pilot we plan to use the senior person simulator, which will be executed in a Raspberry Pi and connected to the Telefónica's testbed. Every time this simulator generates data it will trigger the execution of an EDGELESS workflow.

This simulator will emulate the behaviour of an elderly person at home, generating sensor data (with the same format and periodicity as the real sensors). The simulator will have complexity that allows it to emulate sensors following complex rules and with a certain degree of randomness. For the pilot we intend to use not a single instance of the simulator, but several instances, which can be useful to test EDGELESS scalability. The simulator instances will not be completely equal, but instead they will emulate different profiles (different home's distributions, different person's routines, etc.), which will allow us to test the ML algorithms accuracy in different scenarios.

The workflows executed will identify activities and detect anomalies, and then the data will be sent to the Senior Care App. In the senior Care App we'll be able to analyse the data received and validate the correct end-to-end operation of the system.

The end-to-end operation (from the senior person simulator to the Senior Care App) will allow us to evaluate the EDGELESS functionalities and requirements, such as the latency and the connection with external APIs; and also the precision of the Use Case ML algorithms.

4 Conclusion

The deliverable describes the conducted work in WP5 and is the first of three deliverables and represents the foundations in regards to the integration and deployment of Edgeless into the UCs, as well as more in depth definition of the UCs, with some of desired Edgeless features being identified from the UC side.



6 References

"Repositories documentation", GitHub. <https://docs.github.com/en/repositories> (accessed Feb. 13, 2024).

"GitHub Actions documentation", GitHub. <https://docs.github.com/en/actions> (accessed Feb. 13, 2024).

"GitHub Issues documentation", GitHub. <https://docs.github.com/en/issues> (accessed Feb. 13, 2024).

"About releases", GitHub. <https://docs.github.com/en/repositories/releasing-projects-on-github/about-releases> (accessed Feb. 13, 2024).

"About wikis", GitHub. <https://docs.github.com/en/communities/documenting-your-project-with-wikis/about-wikis> (accessed Feb. 13, 2024).

