TALON

Autonomous and Self-organized Artificial Intelligent Orchestrator for a Greener Industry 4.0

Deliverable

D3.6 Overall Architecture & Platform

Actual submission date: 30/06/2025

Funded by
the European Union

Project Number:	101070181		
Project Acronym:	TALON		
Project Title:	Autonomous and Self-organized Artificial Intelligent Orchestrator for a Greener Industry 4.0		
Start date:	October 1st, 2022	**Duration:**	36 months
D3.6 Overall Architecture & Platform			
Work Package:	WP3		
Lead partner:	ENG		
Author(s):	Sergio Comella (ENG), Sophia Karagiorgou (UBI), Ons Aouedi (UL), Maria Boluda (UPV)		
Reviewers:	Li Vlad (KU), Stylianos Trevlakis (IC)		
Due date:	30/06/2025		
Deliverable Type:	DEM, Pilot, Prototype	**Dissemination Level:**	PU
Version number:	1.0		

Revision History

Version	Date	Author	Description
0.1	01/04/2025	ENG	Toc Definition
0.2	15/05/2025	UBI	Contribution in Section 2.2
0.3	20/05/25	ENG	Contribution in Section 2.3 and Chapter 1
0.4	28/05/2025	UPV	Contribution in Section 2.6 and 3.2
0.5	11/06/2025	UL	Contribution in Section 2.5
0.6	12/06/2025	UBI, ENG	Contribution finalisation
0.7	17/06/2025	ENG	Document preparation for internal review
0.8	23/06/2025	ENG	Edit after Internal Review
0.9	24/06/2025	MINDS	Quality Check
1.0	27/06/2025	ENG	Final coordinator review before submission

Table of Contents

List of Figures

List of Tables

Definitions and Acronyms

ACLs	*Access Control Lists*
API	*Application Programming Interface*
CA	*Consortium Agreement*
CNI	*Container Network Interface*
CPU	*Central Processing Unit*
DAG	*Directed Acyclic Graph*
DoA	*Description of Action*
DPDK	*Data Plane Development Kit*
DSL	*Domain Specific Language*
E2C	*Edge-to-Cloud*
eBPF	*extended Berkeley Packet Filter*
EC	*European Commission*
EU	*European Union*
GA	*Grant Agreement*
JIT	*Just-In-Time*
KSM	*Kube-state-metrics*
LSTM	*Long Short-Term Memory*
ML	*Machine Learning*
NFV	*Network Functions Virtualisation*
NG-SDN	*Next Generation Software Defined Network*
NLAs	*Node Level Agents*
NN	*Neural Network*
OVN	*Open Virtual Network*
OVS	*Open vSwitch*
PC	*Project Coordinator*
PPE	*Personal Protection Equipment*
QoS	*Quality of Service*
RAM	*Random Access Memory*
RMSE	*Root Mean Square Error*
SLOs	*Service Level Objectives*
TC	*Technical Coordinator*
TrL	*Trust Level*
UCs	*Use Cases*
WP	*Work Package*
XAI	*Explainable Artificial Intelligence*

Disclaimer

Executive Summary

Deliverable D3.6 represents the final technical output of the TALON project, consolidating all the architectural designs, system integrations, and technological innovations developed over the course of the initiative. It provides a comprehensive, integrated and validated platform for autonomous AI orchestration across edge-to-cloud infrastructures, designed for Industry 4.0 environments where adaptability, efficiency and resilience are paramount. The TALON platform integrates a variety of technologies, including Kubernetes-based orchestration, LSTM-powered decision engines, federated learning, explainable AI (XAI) and blockchain, to create a self-managing, zero-touch system. The architecture enables components such as the Service Orchestrator, Network Intelligence, Resource Manager, and Self-Healing modules to interact seamlessly. These modules work together to monitor performance, enforce service-level objectives (SLOs), detect anomalies, and execute automated remediation actions with minimal human intervention.

This deliverable finalises the platform's entire feedback loop, covering everything from data ingestion and AI model training to deployment, monitoring, adaptation, and system recovery. Explainability checkpoints and Trust Levels (TrLs) ensure fairness, reliability and transparency are embedded throughout the pipeline, while Green-AI principles and benchmarking tools support energy-efficient optimisation at every stage. Blockchain integration provides traceability and auditability of AI decisions and updates, thereby reinforcing trust in distributed, dynamic settings. The platform has been deployed and tested in real-world pilot settings, demonstrating strong autonomous capabilities. It managed workload distribution in real time, triggered alerts based on live system data and corrected resource or application issues without manual intervention. These results demonstrate that the system is technically mature, functionally complete and well aligned with the TALON project's overall goals. As the final deliverable, D3.6 documents the architecture in its most complete form, reflecting on lessons learned and identifying opportunities for future improvement. Although the current system performs well under pilot conditions, it still relies on rule-based remediation and inference-only AI models. Planned developments include the addition of predictive intelligence, support for online and continual learning, integration of event-driven scalability (e.g. via message queues) and enhancement of the user experience through intelligent guidance in SLO configuration and orchestration oversight.

Ultimately, D3.6 encapsulates the vision and achievements of the TALON project. It provides a future-proof, modular platform that can support next-generation industrial AI systems which are self-organising, explainable, energy-efficient, secure and adaptive. With this foundation in place, TALON is ready for real-world adoption, further development and long-term exploitation across diverse industrial sectors.

I Introduction

1.1 Scope and Objectives

Deliverable D3.6 “Overall Architecture & Platform” provides a comprehensive, end-to-end specification and integration of the TALON system’s architectural components, workflows and deployment paradigms. Building on the conceptual blueprint defined in D3.1 and the initial orchestration prototype presented in D3.3, this deliverable will (i) consolidate and refine the Data-Plane and Control-Plane component designs—including the Authentication & Authorization, Data Anonymization, Distributed Ledger, Anomaly Detection, Application Lifecycle Manager, NG-SDN/Distributed Intelligence, Orchestration, Smart Policy Manager, Monitoring & Aggregation, AI-Model Training & SLO, Self-Healing, Transfer-Learning, DataOps, Digital Twin, XAI, Polyglot Data Management and Visualization Dashboard modules—(ii) specify their interfaces and deployment patterns across the Edge-to-Cloud continuum.

D3.6 ensures that TALON’s zero-touch, energy-efficient, secure, explainable and reusable AI orchestration platform is fully specified for implementation, pilot integration and large-scale deployment.

1.2 Relation to other work packages, tasks and deliverables

Deliverable D3.6 synthesises the architectural work of WP3, building directly on the system and user requirements defined in D2.1 (Use Case, KPIs, Requirements, Specification, Slices & Technology Enablers Definition Report) and the Experimental Verified & Optimized AI Theoretical Framework of D2.2. Within WP3, it unifies and refines the outputs from D3.1 (Architecture & Platform Design Blueprint), D3.2 (NG-SDN & Distributed Intelligence Functions Toolkit), D3.3 (Initial Overall E2C AI & Resource Orchestrator) and D3.4 (Self-Healing, Self-Recovery & Self-Correcting Mechanisms Toolkit), weaving these elements into a cohesive platform specification.

D3.6 then becomes the reference for TALON’s core services, interfaces and APIs. WP4 embeds this blueprint into its XAI modules and blockchain-based security toolbox, while WP5 leverages it to orchestrate CI/CD pipelines, pilot deployments and KPI evaluations in D5.1–D5.4. The platform architecture in D3.6 also underpins WP6’s dissemination, standardisation and business planning activities—shaping market positioning, standards roadmaps and go-to-market strategies in D6.1 and D6.2. By integrating requirements, architectural design, AI-driven orchestration logic and self-healing mechanisms, D3.6 ensures a seamless transition from research through integration, validation and exploitation across the entire TALON project.

1.3 Document Structure

This deliverable is organised into four main sections, each building progressively toward a complete specification of the TALON platform.

Section 1 – Introduction (i.e. this section) sets out the scope and objectives of D3.6, explains how it builds on and relates to other work packages, tasks and deliverables, and previews the contents of the report.

Section 2 – System Architecture presents the end-to-end design of the Edge-to-Cloud AI & Resource Orchestrator, detailing its principal building blocks—the Service Orchestrator, Network Intelligence—and their key features and interactions.

Section 3 – Technical Specifications describes the underlying technical decisions that guide the orchestrator’s implementation and operation, including our model taxonomy and the performance metrics we employ to assess each component.

Section 4 – Conclusion summarises the deliverable's main achievements and outlines next steps, potential enhancements and areas for further research and development.

2 Final Architecture Design

2.1 Evolution from initial design

The TALON architecture serves as a comprehensive, end-to-end, AI-fuelled framework designed to facilitate the edge-to-cloud applications lifecycle through a well-orchestrated pipeline of technical components. Its modular design ensures a comprehensive workflow, from loading user-provided datasets and AI models to delivering performance insights, resource metrics, and actionable feedback. TALON's technical components are organised into interconnected tasks (outlined in the TALON workplan), each having a specific technical role in achieving performance, energy efficiency, and runtime adaptations.

At a high level, the user interacts with TALON by monitoring edge-to-cloud metrics collected at the network and edge-cloud level, has access to DataOps for detection of inconsistencies and curation, while defining service-level objectives (SLOs) such as AI model accuracy, energy reduction, resource optimisations. TALON automatically processes these inputs across its pipelines, combining advanced AI and Machine Learning (ML) techniques, federated, few-/zero-shot learning, orchestration tools, eXplainable AI (XAI), and continuous feedback mechanisms to deliver optimised results.

As illustrated in Figure 1, we pass through the architecture from top to bottom and right to left, starting with the user at the top. The user's data and models flow systematically and logically through TALON's core components, including checkpoints for data and AI models assessment for abnormal behaviour and inconsistencies. Then, data undergoes curation and enhancement processes, such as balancing, bias removal, imputation, feature engineering, etc. via the DataOps.

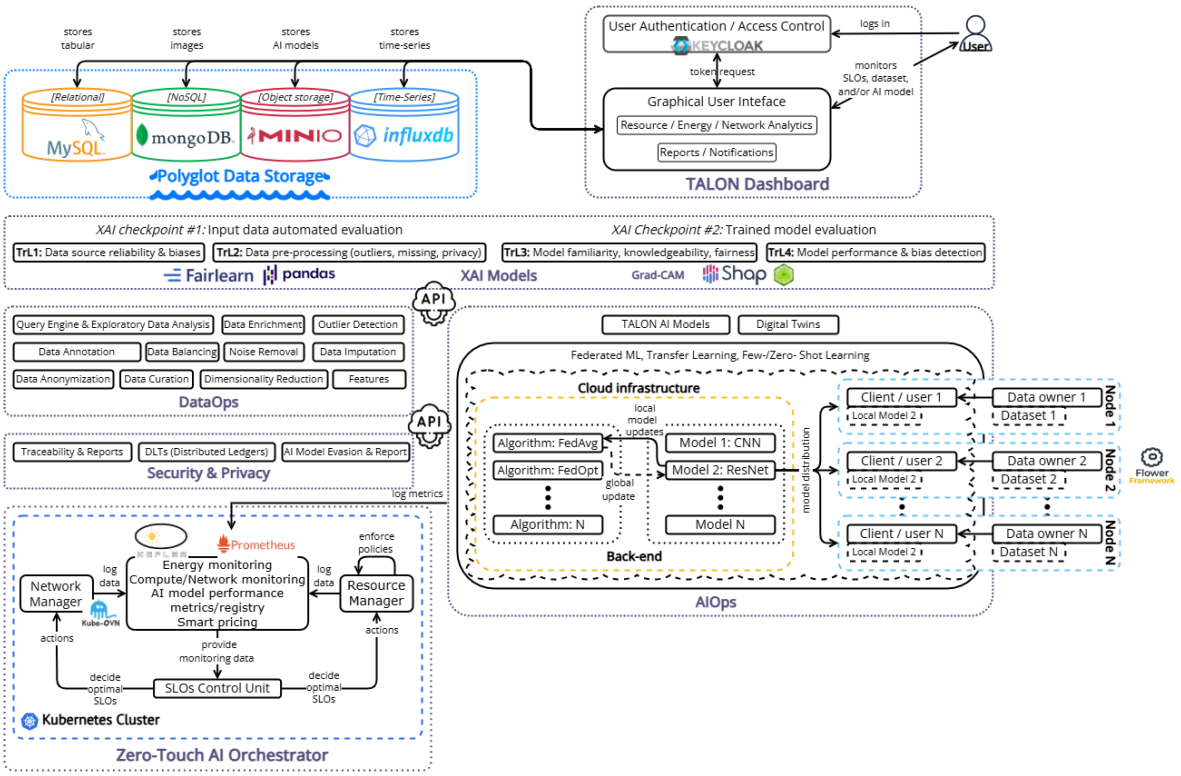

Figure 1. Updated TALON Architecture.

To learn patterns, then data passes through centralised and federated, few-/zero-shot learning and intensive benchmarking to derive insights about green-AI model training instantiated through AIOps. The compute and network resources allocated for all TALON tasks are logged and monitored to

further learn patterns about intelligent and energy-efficient orchestration, resulting in edge-to-cloud policy enforcement according to the SLOs and execution context, optimised AI models, energy-aware deployments and task offloading, as well as actionable recommendations for the user.

The TALON workflow begins with the user, who interacts with the TALON Dashboard. This provides an intuitive Graphical User Interface (GUI) to transform datasets, monitor AI models, define SLOs, monitor workflows, and evaluate results such as performance metrics, energy usage, and other edge-to-cloud workload insights. Secure login is ensured through Keycloak-based user authentication.

The Polyglot Data Storage serves as the centralised storage for all user-input data, derived outputs, trained models, and monitoring logs. It integrates data persistence frameworks like MySQL, MongoDB, MinIO, and InfluxDB to handle diverse data formats (e.g., tabular data, time-series data, trained AI models, and images).

User-provided datasets are ingested and prepared for further processing via DataOps. Before being processed, data passes quality and validation assessment to identify biases, outliers, missing values, inconsistencies in order to apply necessary preprocessing steps such as data cleansing, normalisation and resizing, anonymisation, feature engineering, data enrichment and annotation. TALON ensures that input data meets reliability standards via XAI using tools such as Fairlearn and Pandas (XAI Checkpoint #1). For example, a user may upload a dataset containing imbalanced samples (e.g., biased labels for image classification). The XAI Checkpoint #1 identifies these issues and recommends data rebalancing strategies, while DataOps curate the data and enforce the respective cleansing policy to ensure the dataset is fit for model training. In this component, Trust Levels (TrLs) are introduced as a measure of data reliability and fairness. The XAI Checkpoint #1 includes TrL1 – TrL2, addressing fairness, bias, missing values, errors and inconsistencies, while it passes findings to DataOps to take an action and curate the data.

To address data privacy and decentralised data challenges, TALON implements Federated Learning (FL). This component enables collaborative model training across distributed datasets owned by multiple clients/users. Client Nodes train local model updates using their private datasets. TALON aggregates these local model updates using specialised algorithms (e.g., FedAvg [1] or FedOpt [2]) into a globally updated optimised model.

TALON also incorporates the benchmarking tool of the AI Theoretical Framework [3] which optimises both the energy efficiency and performance of AI models during training and deployment. The tool logs the behaviour of various AI models trained with the TALON datasets and focuses on minimising resource consumption through techniques like energy-efficient hyperparameter tuning, optimal model selection for edge or cloud contexts, and the continuous optimisation of training pipelines. By integrating Green-AI principles, TALON ensures that AI models can be trained effectively without excessive energy or resource usage, thereby reducing the environmental impact of computationally intensive tasks.

The TALON Zero-touch AI Orchestrator, powered by Kubernetes, acts as the brain behind the operation, bringing together all the edge-to-cloud workloads and ensuring they work in harmony to maintain the clusters robust. The orchestrator coordinates edge-to-cloud workflows, takes decisions about runtime adaptations on resources, task or data offloading according to the collected monitoring metrics and remaining battery life / energy supply. It ensures efficient scaling, monitoring, scheduling, and execution of processes while optimising compute and network resources through tools like Prometheus and Kepler. TALON integrates a Network Intelligence, Resource Manager and Service Orchestrator to track system performance metrics (energy consumption, Central Processing Unit (CPU) / Random Access Memory (RAM), bandwidth, etc.) and enforce optimization policies. If a task exceeds user-defined SLOs (e.g., high resource consumption), TALON can dynamically adjust

configurations, like scaling down compute and/or network resources or proposing alternative deployments through task offloading.

After a model has been trained and optimised, XAI Checkpoint #2, utilising tools such as SHapley Additive exPlanations (SHAP), Grad-CAM or LIME, evaluates the AI model during the training and validation stages. This checkpoint ensures and verifies that the model is both trustworthy and explainable (via TrL3 – TrL4), addressing fidelity, confidence and trust to the AI model decision-making mechanism.

In order to enhance transparency, security, and traceability, TALON incorporates Blockchain technology within its framework. Blockchain serves as an immutable and wisely deployed energy-efficient ledger, functioning as a tamper-evident, auditable record system designed to minimize environmental impact. Notably, this is accomplished by substituting proof-of-work with energy-efficient consensus mechanisms for data or AI model updates, such as proof-of-stake, embedding real-time energy-use metrics in each block, automatically retiring corresponding carbon credits via smart contracts, and mandating that validator nodes operate on verifiable renewable energy, with on-chain oracles exclusively reporting consumption and offsets. This methodology ensures that only updates across distributed environments are subject to audit for evasion, consistency, and integrity. Given these components, TALON provides an adaptable, efficient, environmentally friendly, and user-centric AI development environment that effectively balances performance, resource optimization, and fairness.

2.2 Limits and future considerations

The TALON platform presents an innovative approach to green-AI edge-to-cloud deployments, addressing critical aspects like data privacy, explainability, and adaptability. However, like any advanced system, it operates within certain current limits, offers valuable lessons learned from its design, and points towards considerations for further development. We highlight below some considerations and plans for future exploitation:

- The continuous monitoring, runtime adaptation, and optimisation by the Green-AI frameworks and self-evolving zero-touch AI orchestrator, while beneficial, inherently introduce a certain level of computational and operational overhead. It also required historical monitoring data which in some cases are not made available by the organisations. The balance between optimisation benefits and the cost of this overhead needs careful management. Optimising energy and resource usage has been considered as integral driver to sustainable AI development.
- While Blockchain enhances transparency, security, and traceability, its implementation (immutable ledger, tracking model updates) may introduce overhead in terms of storage, computational requirements, and latency, which might impact performance for very high-frequency updates. To overcome this shortcoming, in TALON we decided to only log the learnt AI model weights and minimise to the possible minimum the frequent updates. An abnormal behaviour / unauthorised access to an update is reported back to the TALON system as an incident of AI model evasion.
- The explicit inclusion of XAI Checkpoint #1 (Fairlearn/Pandas for bias/quality checks) and DataOps at the data ingestion stage underscores the critical lesson that the reliability and fairness of input data directly dictate the trustworthiness and performance of downstream AI models. Introducing actionable Trust Levels (TrLs) since M18 is a key learning.
- The enhanced user guidance and AI-assisted SLOs definition should incorporate more intelligent tools within the TALON Dashboard to guide users in defining optimal SLOs,

potentially recommending them based on dataset characteristics, AI model type / dimension, and available resources, simplifying the initial setup and in-the-course adaptations.

- The automated policy generation for the orchestrator ecosystem (e.g., smart pricing, energy allocation, incentives for optimal deployments considering off-peak hours, renewables, etc.) drives the potential for the TALON system to go beyond a decision-making platform and automatically generate and enforce network or resource policies based on dynamically calculated cost, Trust Levels (TrLs), and workloads further automating the self-managing aspect of the platform.

2.3 Network Intelligence Final Architecture

2.3.1 Components Overview

Network Functions Virtualisation (NFV) and Software Defined Networking (SDN) have transformed the framework for deploying Edge-to-Cloud (E2C) services. On one hand, vast amounts of data are produced every second by interconnected software and hardware services. On the other hand, data collection and data quality are often secondary concerns. Feature engineering algorithms are currently applied in modern SDN infrastructures for optimising network functions and injecting intelligence at the virtual network functions. High-quality data ensure that SDN controllers can make precise adjustments to network configurations, optimise performance, and proactively respond to evolving conditions. Moreover, a well-crafted set of features not only enhances the accuracy of predictive models but also facilitates intelligent decision-making within a dynamic network environment.

SDN and Artificial Intelligence (AI) are two distinct but increasingly interconnected technologies that play a crucial role in modern network management and optimisation. AI can play a pivotal role in automating network policy management and monitoring by providing intelligent and adaptive solutions to emerging challenges in E2C applications. The applicability of automated decisions via network policies is made by analysing and learning motifs on network traffic patterns. The real time monitoring of network metrics allows the derivation of behavioural patterns for diverse applications, fine-tuning their thresholds, optimising the network, identifying anomalies, performing predictive maintenance, and predicting potential security threats. As a result, the interplay between network management mechanisms, AI, and network policies enhances the availability, security, stability, and scalability of the edge-to-cloud ecosystem. This proactive approach enables the development of dynamic network policies that can autonomously adjust to evolving issues (e.g., load balancing in response to traffic bursts), thereby improving the overall resilience of the network infrastructure. Additionally, AI Models for network intelligence contribute to the creation of more sophisticated control mechanisms that fine-tune their network policies based on network / application / pod behaviour, infrastructure's environmental characteristics, and contextual information.

The components composing the Network Intelligence include a Kubernetes cluster of nodes built on top of Kube-Ovn network interface, both allowing for managing network resources and virtualising network functions. This setup utilizes most of the networking functionalities, including Quality-of-Service (QoS) management. The entire infrastructure (nodes, pods, and services) is continuously monitored by utilizing different agents collecting metrics, such as Prometheus, and Kepler to measure different behaviours of deployed pods, applications and resources. The data logs are further stored in an InfluxDB database for long-term data persistence.

The deployment of these frameworks and the automatic metrics collection process are responsible for populating measurements which are then fed to AI models for learning application patterns. The results derived from the AI model training serve to meet specific QoS requirements (e.g., high availability, high bandwidth, etc.) and Service Level Objectives (SLOs) (e.g., latency below 5ms). The

monitoring metrics, SLOs and the learnt patterns from AI model training are combined to enforce network rules / policies via YAML files when an event occurs to adapt network resources at runtime. Overall, combining the deployed frameworks and learning pods / application patterns on logged data, the TALON Network Intelligence can dynamically adjust its behaviour, configuration, and resource allocation while it is actively running, in response to changing conditions, demands, or failures. The goal of these runtime adaptations is typically to optimize performance, maintain reliability, enhance security, and improve efficiency without requiring a complete system restart or manual intervention.

The final architecture of the Network Intelligence is depicted in Figure 2. The core functionality of this component is to manage virtual overlay networks between edge and cloud. The input to these overlay networks consists of network rules / policies that dynamically update routing or replace a pod within the cluster at runtime through policy-enforcement. The component continuously feeds itself with new measurements to optimize the deployments based on the target QoS and SLOs.

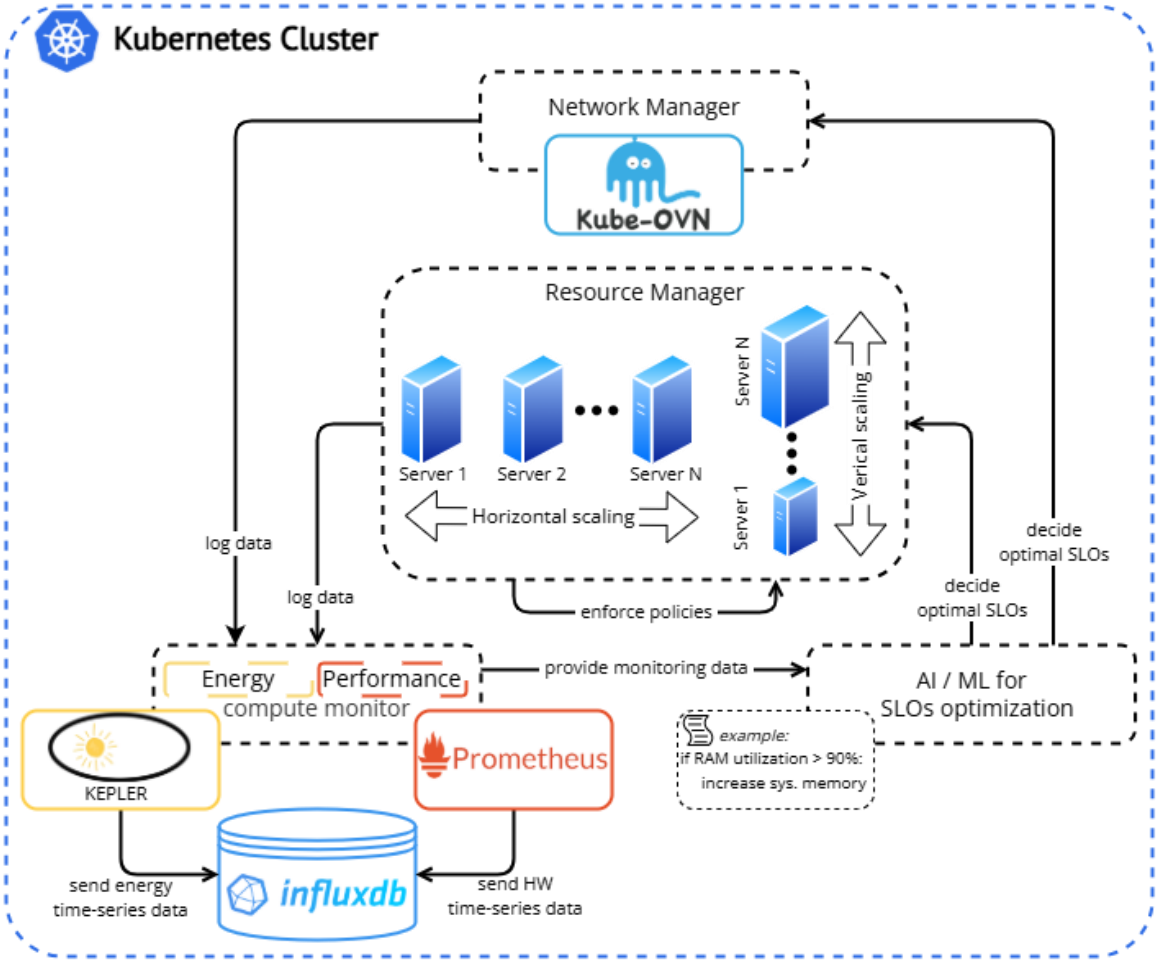

Figure 2. Network Intelligence Final Architecture.

2.3.2 Limits and Future considerations

The effectiveness of the Network Intelligence, especially the AI models and the resulting network adaptations, is heavily dependent on the quality of the collected data. If the data fed to the AI models is incomplete, or inaccurate, the learned patterns and the subsequent network policies will be suboptimal or even detrimental. Also, while the Network Intelligence aims for runtime adaptation, there will inherently be a latency (e.g., 2-3 seconds, acceptable because most network degradations and path shifts develop on the order of seconds, so acting within that window prevents any noticeable

impact) between the collection and processing of metrics, the occurrence of an event, and the enforcement of new policies. We clarify here that AI models do not introduce latency because the optimal metric or SLO is instantly decided via an inference task. However, this latency from the perception of the event to the enforcement of the policy might be critical for highly dynamic scenarios requiring immediate responses.

For the future, we plan to explore more sophisticated and automated feature engineering techniques that can dynamically identify relevant features based on evolving network conditions and application behaviours to enhance the AI's learning capabilities. Lastly, we plan to **expand the scope of monitored metrics to include more granular network performance indicators,** application-level KPIs, user experience metrics, and security-related events to provide a richer context for AI-driven decision-making. Implementing advanced observability tools beyond basic metric collection will be crucial for understanding complex system behaviours.

2.4 Service Orchestrator Final Architecture

2.4.1 Components Overview

In this section, is described the **Service Orchestrator**, TALON framework's **engine** that seamlessly converts high-level alerts into automated remediation steps executed via Kubernetes APIs [4]. Sitting **between** the monitoring layer and the Resource Manager, it ensures that any pod exceeding defined thresholds for **memory**, **CPU**, **data** or **task load** is detected, assessed, and healed **without manual intervention**.

The Service Orchestrator sits at the heart of TALON, acting as the intelligent engine that transforms high-level alerts into fully automated remediation workflows via Kubernetes. It bridges the monitoring layer and the Resource Manager to detect, analyze, and heal any pod breaching its CPU, memory, I/O or workload thresholds—without human intervention.

When Prometheus AlertManager emits an alert, a lightweight FastAPI webhook captures the JSON payload (including namespace and pod identifiers), logs it, and forwards it to the E2C Orchestrator. The orchestrator retrieves a list of metrics—which it ingests into the UL-developed Resource Manager—for processing; the Resource Manager then returns a scale/offload policy to be applied. These live readings are enriched by historical performance profiles in InfluxDB and real-time metrics from Prometheus, providing a holistic, time-aware snapshot of each pod's behavior [5].

Next, the orchestrator hands this policy back and executes the chosen remediation action—whether in-place CPU/memory tuning, replica scaling, pod eviction for rescheduling, data offload, or task rerouting—scored against latency impact, throughput change, and energy efficiency, all while ensuring compliance with TALON's Service Level Objectives (SLOs). Once an optimal action is chosen, the orchestrator invokes its core decision logic in perform_action(namespace, prediction, pod_name). If the returned magnitude equals **1.0**, no change is made.

Otherwise, the policy string is split on underscores to dispatch to the appropriate helper method described in Table 1 below.

Policy	Method	Kubernetes API Call	Description
cpu_scale_up/down	adjust_cpu	patch_namespaced_pod	Reads current CPU requests/limits, applies the factor, and patches the pod spec.
memory_scale_up/down	adjust_memory	patch_namespaced_pod	Reads current memory requests/limits, applies the factor, and patches the pod spec.

task_offloading	task_offloading	Delete + Create	Annotates the pod and move the task to another node.
data_offloading	data_offloading	patch_namespaced_pod (annotation)	Annotates the pod to trigger data offload workflows.

Table 1. Policies performed by E2C Orchestrator.

The scaling helpers (scale_cpu, scale_memory) parse unit suffixes (e.g. “m”, “Mi”, “Gi”), convert values to base units, multiply by the factor, and reassemble the properly suffixed string. All API calls, responses and any exceptions are logged with full context (exc_info=True), ensuring complete observability [6]. Once an optimal action is chosen, the orchestrator issues the corresponding Kubernetes API call—patching resource limits, scaling deployments, evicting pods, or triggering offload routines—and marks the operation as in-progress. All state transitions and API responses are streamed back into the webhook’s log store and emitted as custom Prometheus metrics. If the same alert recurs within a brief cool-down interval, the orchestrator transparently escalates to the next-best policy, closing a continuous self-healing loop that dynamically adapts to evolving cluster conditions.

Resource limits and requests live in a Pod’s spec (`spec.containers[*].resources`), and—as a rule—once a Pod is running you cannot mutate those values in place. Any change to CPU/memory requests or limits requires the Pod (i.e. the container) to be restarted (recreated) before the new settings take effect.

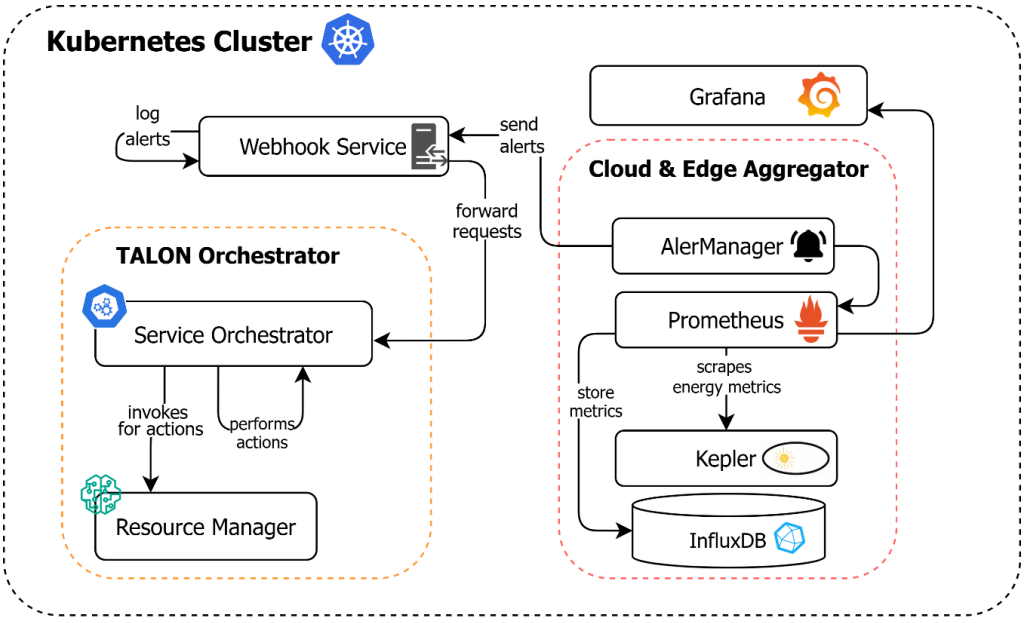

Figure 3. Service Orchestrator flow

Figure 3 illustrates this end-to-end flow: from **AlertManager** through the webhook interface, into the orchestrator’s decision pipeline, and onward to the **Resource Manager** and Kubernetes API. By decoupling alert ingestion, decision logic and execution—and enabling horizontal scaling of orchestrator replicas—the Service Orchestrator delivers a **robust**, **extensible** foundation for TALON’s real-time resource optimization.

The Service Orchestrator embodies TALON’s commitment to **fully automated, policy-driven** resource management: from the moment Prometheus alerts arrive until Kubernetes API calls enact the chosen remediation, every step is instrumented for observability, resilience and scalability. By combining real-time metrics, historical performance profiles and an AI-powered policy engine, it ensures that pods are scaled, rescheduled or offloaded in a way that consistently upholds Service Level Objectives. The modular design—decoupling alert ingestion, decision logic and execution—

allows orchestrator replicas to grow with demand, while the continuous self-healing loop adapts to changing workloads without human intervention [7]. Altogether, this section has shown how the Service Orchestrator provides a robust, extensible foundation for TALON's real-time resource optimization.

2.4.2 Limits and Future considerations

The Service Orchestrator's current architecture handles alerts **immediately** as they arrive—without any **buffering** or **message-queuing** layer—which means that a sudden surge of fired alerts can exhaust thread pools or overwhelm the **FastAPI** webhook, resulting in increased latency or even dropped remediation requests. Because every decision path invokes the external AI policy endpoint, any slowdown or outage there becomes a **single point of failure**: the entire self-healing pipeline stalls until the policy service recovers. At the same time, relying on a static **"cool-down"** interval for re-alert escalation fails to account for workload variability—temporary traffic spikes may be misinterpreted as repeated violations, triggering unnecessary, more aggressive policies.

To transform TALON's orchestrator into a truly **resilient**, **scalable** and **context-aware** engine, we can enrich the design by introducing a lightweight **event-streaming** layer (e.g., Kafka or RabbitMQ) between AlertManager and the orchestrator to absorb bursts and guarantee delivery, allowing replicas to scale elastically based on queue depth. Embedding a **fallback policy**—for example, a simple threshold-based scaler—or applying a **circuit-breaker** around the AI service ensures that, even during transient network failures or service outages, critical remediation still proceeds. Replacing the fixed cool-down with **adaptive**, history-informed intervals would distinguish true repeat violations from benign resource blips, minimizing unnecessary escalations.

2.5 Resource Manager Final Architecture

2.5.1 Components Overview

This section presents the finalized architecture of the TALON Edge-to-Cloud (E2C) Resource Manager, designed for dynamic and intelligent task orchestration. Its architecture integrates AI-driven decision-making with real-time telemetry analysis, enabling scalability and energy efficiency, and is aligned with Industry 5.0 principles.

Key components include:

- Data Collector and Monitor: Continuously captures fine-grained metrics such as CPU usage, memory load, energy consumption, and system-level parameters from edge and cloud nodes. It leverages Kepler-compatible tools for accurate energy profiling.
- LSTM-based Policy Decision Engine: A central AI module built on a Multi-Task LSTM architecture described in Deliverable D3.3, trained to perform both classification (policy selection) and regression (magnitude estimation). An attention mechanism is integrated to highlight the most critical time steps for decision-making. This dual-task structure enhances efficiency, robustness, and responsiveness of the orchestration process. This component processes time-series input—including workload intensity, energy consumption, and temporal signals—to predict (Figure 4)
    - The optimal orchestration policy (e.g., scale up/down, task or data offloading, no action).
    - The required magnitude of the chosen action (e.g., how much to scale or offload).

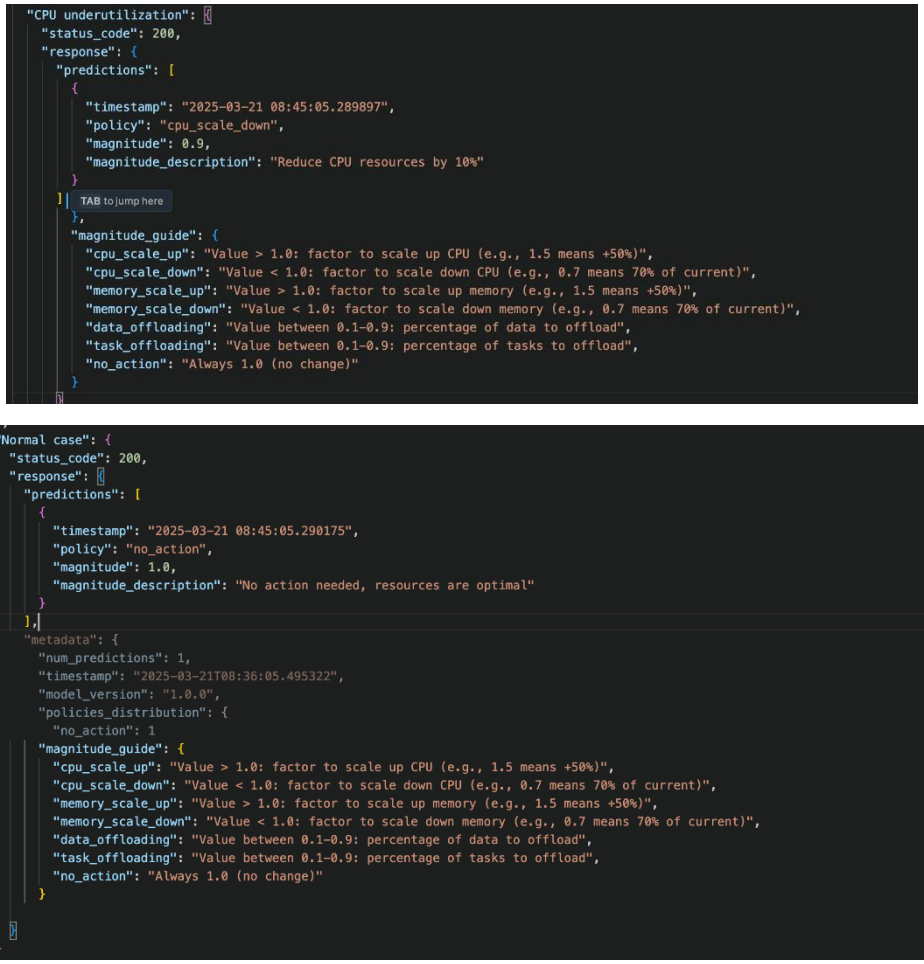

Figure 4. LSTM-based Policy Decision Engine.

2.5.2 Limits and Future considerations

While the Resource Manager demonstrates strong potential, some limitations merit attention and will be addressed in future work, such as the LSTM-based engine operating in inference-only mode, without mechanisms for continuous online learning or retraining. This can lead to model drift over time as workloads or environments evolve.

2.6 Self-Healing, Recovery & Correcting Final Architecture

2.6.1 Components Overview

Modern distributed infrastructures demand resilience mechanisms autonomously capable of detecting, mitigating, and recovering from disruptions without human intervention. The self-healing and self-correcting architecture developed addresses this need by integrating real-time monitoring, alerts and automated response workflows within a microservices-based environment.

The architecture relies on **K3s**-based Kubernetes cluster orchestrated through **Rancher** and is enhanced with monitoring and control tools such as **Prometheus**, **Grafana** and **Node-RED**. As shown in Figure 5, the control plane node resides in the cloud and governs multiple worker nodes distributed across both cloud and edge environments. This node is hosted on the virtual machine with the IP **10.1.6.107**. Prometheus, Grafana, and Node-RED are deployed as microservices in dedicated pods (1AC, 1AB, and 1AA, respectively) running on a cloud worker node (Worker 1), which is hosted on a separate virtual machine with IP **10.1.6.111**. Application workloads are distributed across edge workers (Workers 3, 4, and 5).

Together, these components create a closed feedback loop that ensures system stability and service continuity. Prometheus continuously scrapes metrics from pods, nodes, and services, storing them as time series. These metrics are visualised in Grafana dashboards, where alert rules are configured to detect anomalies or threshold violations. When an alert is triggered, it is routed to Node-RED via webhooks, where a predefined rule-based flow evaluates the situation and initiates corrective actions

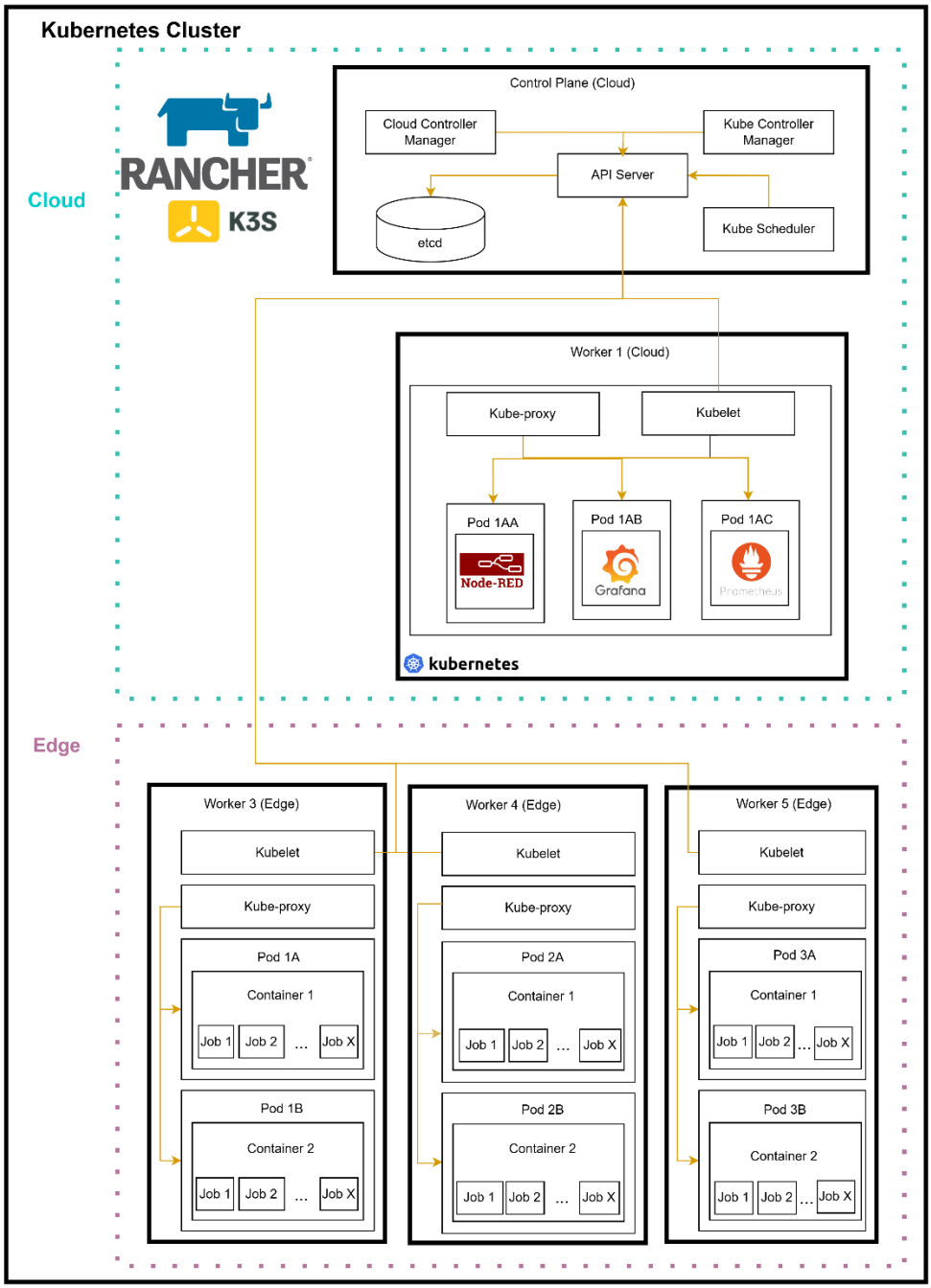

Figure 5. Self-Healing and Self-Correcting Architecture Overview.

The control logic implemented in Node-RED allows for the execution of granular and deterministic workflows, meaning that the system behaves in a predictable manner for given input conditions and is composed of small, well-defined steps that provide fine-grained control over each stage of the corrective process. These workflows must be preconfigured; however, once deployed, the reaction is fully automated, no human intervention is required to detect the issue or execute the corrective action.

For instance, when CPU usage exceeds a critical threshold, Node-RED issue API calls to scale down pods, taints overloaded nodes or restarts malfunctioning containers. These actions are transmitted to the Kubernetes API, often through Rancher's authentication layer, ensuring secure, traceable and consistent enforcement of decisions. At TALON, the necessary Node-RED flows have been developed and provided as part of the system implementation, ensuring that the corrective logic is functional from the outset and does not require users to define the rules manually.

The architecture supports both vertical and horizontal remediation strategies. Vertical remediation includes adjusting resources or modifying deployment configurations, while horizontal approaches involve redistributing workload, rescheduling pods, or scaling deployments. The use of declarative APIs ensures that these changes are applied consistently and that system state converges towards a stable configuration.

The flows implemented follow a structured pattern that includes alert reception, data extraction, severity evaluation, action preparation, and execution. Debug nodes within Node-RED provide visibility at each stage, allowing traceability and post-mortem analysis of incident responses. These flows are capable of handling multiple events concurrently under normal operating conditions, if hardware resources are not saturated.

The control plane includes core Kubernetes components such as the API server, kube-scheduler, controller managers, and etcd, ensuring full orchestration capabilities. Edge worker nodes run application pods that encapsulate containers and jobs, enabling distributed processing and decentralised management. Communication among all components relies on internal APIs and service discovery mechanisms native to Kubernetes.

Overall, the system operates as an autonomic control loop where infrastructure monitoring and orchestration are tightly coupled. This architecture enables dynamic adaptation to varying workloads, component failures, and performance degradation, ensuring continuous compliance with predefined service level objectives.

Finally, Table 2 is introduced, which summarises the main components of the architecture, including their function, where is deployed within the system and the key features.

Table 2. Summary of the main components.

Component	Function	Deployment	Key Features
Rancher	Kubernetes cluster orchestration and management interface	Cloud control plane	Centralised dashboard, RBAC, authentication, multi-cluster management
Kubernetes	Container orchestration and resource scheduling	Control plane + Edge nodes	Pod lifecycle management, automatic scaling, workload rescheduling
Prometheus	Metric collection and storage	Cloud worker pod (1AC)	Pull-based scraping, time-series storage, PromQL support, exporter integration
Grafana	Metric visualisation and alert generation	Cloud worker pod (1AB)	Custom dashboards, Prometheus integration, webhook alerting to external systems
Node-RED	Execution of rule-based corrective actions	Cloud worker pod (1AA)	Flow-based logic engine, REST API integration, automated remediation workflows
API Server	Entry point to the Kubernetes control plane	Control plane	Receives and authenticates commands from Node-RED, communicates with scheduler and controller manager

2.6.2 Limits and Future considerations

The self-healing and self-correcting architecture developed has been successfully deployed and validated in real industrial conditions within Pilot 2, demonstrating its capability to autonomously detect, mitigate, and recover from operational disruptions with minimal human intervention. Its modular design, based on Kubernetes, Rancher, Prometheus, Grafana and Node-RED, provides a solid and extensible foundation for resilient infrastructure management.

However, one current limitation lies in the rule-based strategy implemented in Node-RED. While this approach has proven highly effective in managing predefined failure scenarios with fast and deterministic responses, it still requires manual design and configuration of flows. As a result, any emergence of new or unforeseen failure patterns necessitates human intervention to adapt or extend the corresponding logic. This constrains the architecture's ability to evolve autonomously in highly dynamic environments.

To overcome this, future improvements will explore the integration of adaptive logic mechanisms that extend the capabilities of the current system. Potential directions include incorporating AI-based recommendation engines, anomaly detection models, and reinforcement learning algorithms that can learn optimal corrective strategies over time. By leveraging historical telemetry and contextual information from the system, these enhancements would enable the architecture to automatically adjust to evolving operational conditions, reduce reliance on static rules, and increase its responsiveness to novel events.

Finally, although latency has been kept minimal in the current implementation, there was no historical performance data available prior to this integration, since no monitoring tools had been in place to capture latency or other metrics. The decision-execution pipeline—from metric collection to action enforcement—still involves an inherent delay. In particular, the system is designed to avoid reacting to transient perturbations: an alert is only triggered when a threshold violation persists for at least one minute, ensuring that the anomaly is real and sustained. While this approach prevents unnecessary corrections, it also means that the system only reacts once the overload or failure has already materialised.

At present, corrective actions are fully automated and executed without human intervention, as defined by preconfigured rules. However, these rules act after the fact, once the issue is already impacting the system. A promising direction for future improvement would be the integration of predictive capabilities, possibly based on AI or statistical pattern detection, that could identify early indicators of potential collapse. This would allow the system to take pre-emptive measures, such as redistributing workloads or scaling resources, before the problem fully manifests, thereby increasing resilience and reducing recovery time.

2.7 Smart Pricing Policies for Non-Commercial Devices Participation

TALON envisions an energy-efficient edge AI network that fully incorporates automation, flexibility, adaptability, programmability, and explainability. To achieve this vision, the TALON system includes a key component: an AI orchestrator designed to boost dynamic scalability by managing diverse computational and communication resources. This orchestrator is intended to improve both data efficiency and energy efficiency (EE) as part of our effort to establish a greener AI network (Project Pillar I).

A central part of our resource management strategy is the Smart Pricing Simulator (SPS), which is integrated into the TALON user interface. The SPS can enhance the allocation of computational resources through four distinct mechanisms:

- Ensuring that charges for using the edge computing network align with the computational requirements of each task.
- Implementing a pay-as-you-go model so that participants know their costs upfront, encouraging them to leverage the energy-efficient edge computing network.
- Facilitating the use of non-commercial (e.g., user-owned) devices by intelligently allocating workloads to underutilized resources.
- Providing an incentivization scheme that allows users to opt in to more energy-efficient options, reinforcing our goal of a greener, more efficient edge computing network.

These Smart Pricing policies, implemented through the SPS, are expected to help us meet the project's ambitious KPI targets. They target at least a 20% improvement in energy efficiency, a 20% improvement in data efficiency, and a 70% or higher participation rate of non-commercial devices in the AI network. The SPS is particularly important for the latter KPI (increasing non-commercial device participation), since leveraging these underutilized resources can significantly improve real-time performance by bringing more intelligence to the network edge.

The SPS has reached TRL 5 and functions as an API, ensuring that is integration ready with any complementary mechanisms, both internal and external. It is capable of receiving real-time data on resource availability and demand patterns and returning optimized pricing strategies that other modules can utilize with minimal manual intervention.

As thoroughly described in D3.5, TALON's SPS incorporates two main pricing schemes: (1) A reservation pricing scheme and (2) a pay-as-you-go pricing scheme. Both of those schemes are combined with incentivization strategies that users can opt-in that are based on energy-efficient computing deployments.

In terms of the **reservation pricing scheme**, the SPS allows users to choose the required edge/cloud computing resources from a list of pre-defined bundles designed to cover a wide range of needs and budgets. More specifically, the SPS currently offers 5 main bundle categories that are designed to align with TALON's pilots in terms of computing requirements. The bundles are as following:

Basic Bundle

- Total vCPU Hours: 100
- Max Simultaneous vCPUs: 2
- RAM (GB): 400
- Discount: 5%
- Description: Ideal for small-scale projects or light parallel tasks.

Standard Bundle

- Total vCPU Hours: 300
- Max Simultaneous vCPUs: 4
- RAM (GB): 1200
- Discount: 10%
- Description: Suitable for medium-scale operations with moderate parallelization.

Professional Bundle

- Total vCPU Hours: 700
- Max Simultaneous vCPUs: 8
- RAM (GB): 2800
- Discount: 15%
- Description: Designed for consistent workloads requiring high parallel processing.

Enterprise Bundle

- Total vCPU Hours: 1500
- Max Simultaneous vCPUs: 16
- RAM (GB): 6000
- Discount: 20%
- Description: Tailored for businesses with complex, parallel tasks.

Ultimate Bundle

- Total vCPU Hours: 3000
- Max Simultaneous vCPUs: 32
- RAM (GB): 12000
- Discount: 25%
- Description: Optimized for enterprises with massive parallel tasks.

To make things more palpable, the enterprise bundle, for example, would be an ideal candidate for heavy model training for Demonstrator #2 I5.0 Automation and Planning, whereas the basic bundle might be more applicable for Demonstrator #1, where UAVs might need to process video data for a short duration.

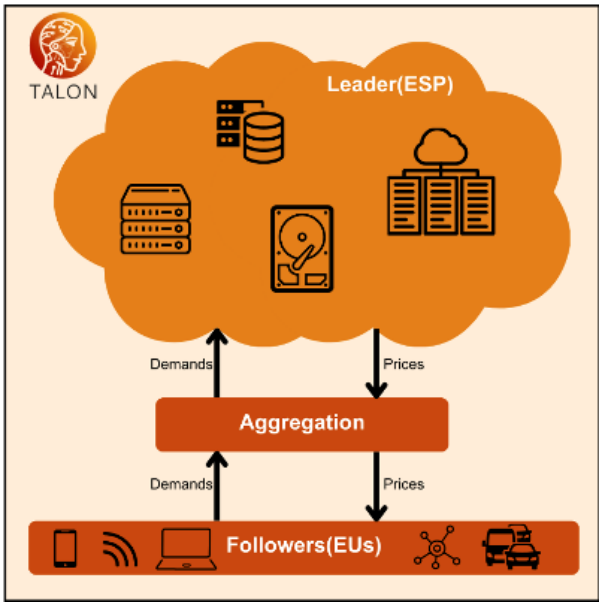

Figure 6. Price-Demand Balance with multiple EUs.

In terms of the **pay-as-you-go pricing scheme**, this approach is ideal for use cases where the precise computational resources required are not clear upfront or are influenced dynamically by external factors. In essence, the SPS allows for real time resource requests (or extensions) based on such dynamic compute demands, while at the same time ensures that the pricing rate for such a dynamic resource allocation is rigidly defined upfront.

To achieve this, the SPS models a Stackelberg game for sequential decision making in the pricing policies. The game is utilized to manage computational resource demand in edge/cloud computing, where the ESP sets per unit or per hour resource prices as the leader and users respond with demand as followers. In essence, the game formulation considers users' reaction to the pricing decisions made by the ESP, based on the average resource price in the market. This helps define users' willingness to pay a certain price for a set of given resources. In simple terms, if the price for certain offerings exceeds the market's balance, then the users are less likely to opt in for the given offer, whereas if it's lower, the ESP profit is less optimal. With the game, the SPS ensures that the optimal balance is found. As shown in Figure 6, the SPS's Stackelberg pricing loop mediates between the ESP, the aggregator, and the end-user devices.

Additionally, a significant advantage of the SPS is that it incorporates a Holt-Winters Seasonal Model for demand prediction, based on which the system can set a fixed price on the pay-as-you-go pricing scheme for a longer duration. Those predictions are essentially taken into consideration within the Stackelberg game and prices are optimized upfront for a long time interval. This means that although users can opt-in for a pay-as-you-go pricing scheme, there are no price fluctuations throughout the time interval that they utilize the edge/cloud computing service. At the time, the ESP maintains optimal profitability as the price is decided based on an aggregation of the predicted demand and its interplay with expected market prices.

For example, if a user requests a pay-as-you-go service, the system (a) forecasts the demand for edge/cloud computing services for the following period, (b) Estimates the average market price based on this demand and historical pricing data, (c) performs a Stackelberg game that utilizes information from (a) and (b) to define an optimal price for the following period. Note that the following period is currently 1 week, but this is adjustable based on the quality and quantity of the historical data (Figure 7 below).

This might be a promising approach for example in Demonstrator 1, where drones might have short flight duration times and require dynamic resources based on the video analytics requirements that a situation dictates.

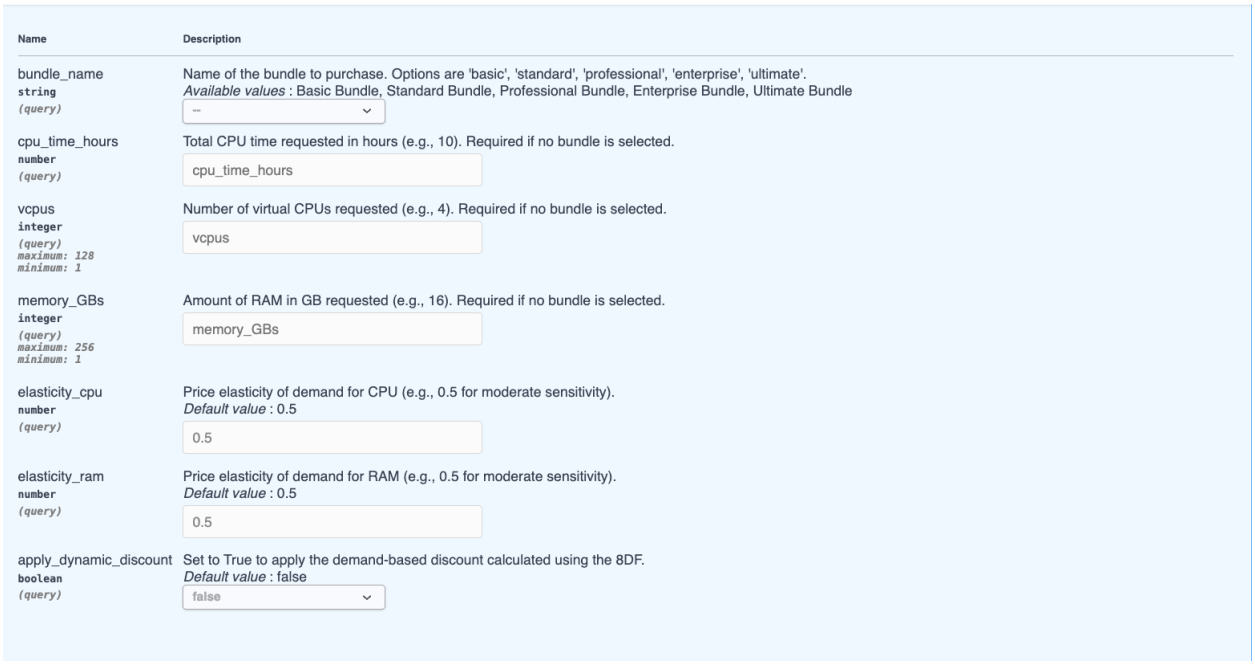

Figure 7. Screenshot from the SPS API Endpoint Parameters.

In terms of user incentivization and energy-efficiency, both the reservation and the pay-as-you-go pricing scheme deploys a discounting factor based on the forecasted demand. In low-demand hours the discount rate can reach up to 20% whereas, in high-demand hours or peak hours there are no discount factors. This approach incentivizes the users to use the cloud/edge computing resources during off-peak hours, ensuring that servers are operated efficiently. It is important to highlight that the reservation pricing scheme offers a bonus discount to users, as it allows ESPs to know their resource utilization rates upfront, therefore managing them more efficiently (Figure 8 below).

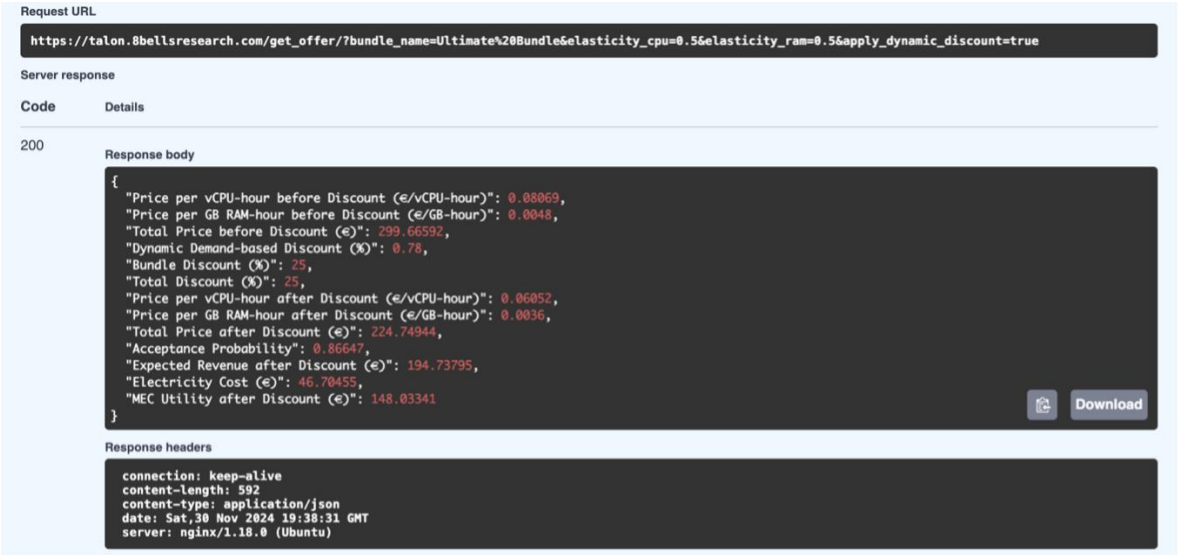

Figure 8. Screenshot from the SPS API endpoint Response

Overall, the SPS may be a promising approach to design policies that bring forward optimal and energy-efficient resource allocation, maximize the ESPs profit, while ensuring that users are receiving the requested service in the best price possible.

3 Autonomous Orchestration and Runtime Operation

3.1 Resource Allocation and Adaptation Mechanisms

In the zero-touch deployment paradigm defined in D3.3, once the Service Orchestrator has dispatched a task to Kubernetes, selecting the optimal configuration for speed, quality, energy or cost, it immediately transitions into an autonomous, self-managing lifecycle. As soon as the pod comes online, Prometheus begins scraping fine-grained telemetry (CPU, memory and energy metrics), and *AlertManager* continuously evaluates these streams against preconfigured rules. A threshold breach (e.g. CPU > 80%, memory headroom < 20%, energy draw > X J/h) triggers an alert webhook to the Orchestrator's */api/alerts* endpoint.

3.1.1 Metrics Collection and Prediction

If the *AlertManager* is triggered and the webhook is invoked, it logs the alert and sends it to the service Orchestrator, as Figure 9 below shows:

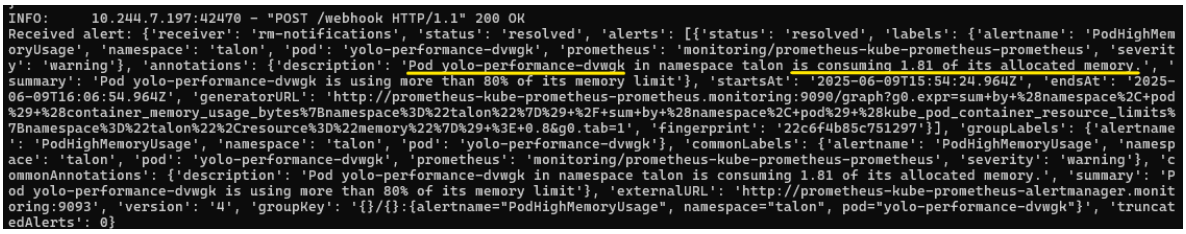

Figure 9. Webhook logs after rule is fired.

When the alert arrives, the Service Orchestrator delegates analysis to its Resource Manager. First, it gathers recent performance data for the impacted pod—pulling CPU utilisation, memory usage, energy metrics (such as DRAM and core joules), and user- versus system-level CPU time over the chosen look-back window. These raw time-series are then normalised into a structured sequence of timestamped feature records. Next, the orchestrator submits this feature set to the Resource Manager's prediction service. The model evaluates the pod's behaviour and recommends a corrective policy, for example, "cpu_scale_up" with a magnitude of 1.5, indicating that the pod's CPU allocation should be increased by 50%. This policy is returned as a simple JSON directive, which the orchestrator interprets and applies automatically to resolve the overload.

Figure 10 below illustrates this flow in action. Starting from the top, an alert from Prometheus *AlertManager* triggers the FastAPI webhook to capture and forward the payload to the Orchestrator. The orchestrator then gathers and normalises the pod's time-series metrics before sending them to the Resource Manager's prediction engine. At the decision point, the model selects one of several remediation paths, such as CPU scaling, memory scaling, or task offload, and the orchestrator applies the chosen action via the Kubernetes API. Finally, monitoring resumes, closing the loop and ensuring continuous, automated remediation.

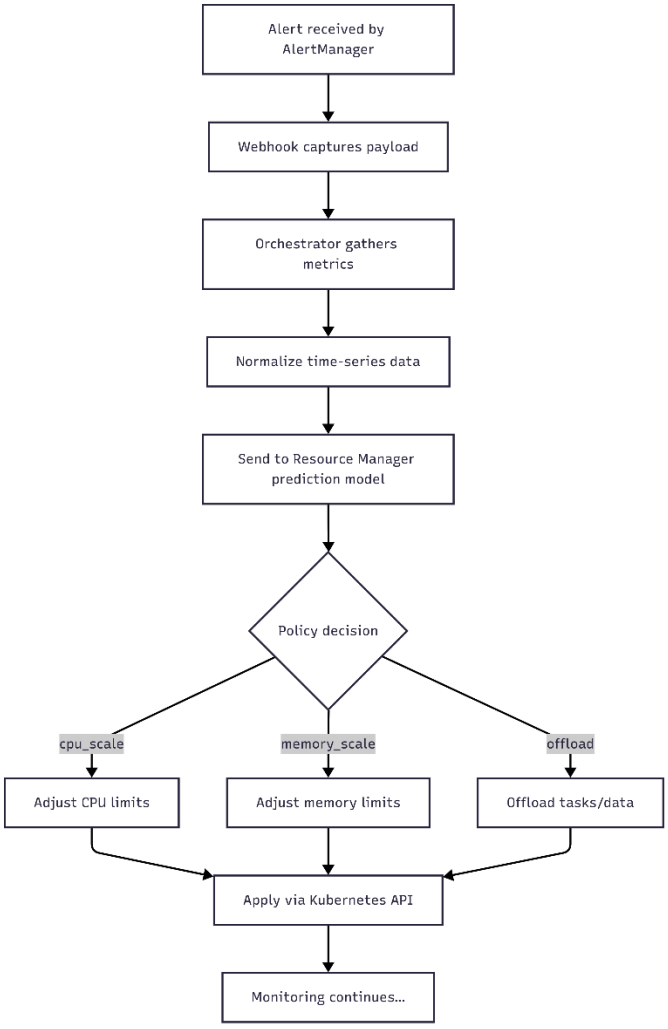

Figure 10. E2C Orchestrator flow.

3.1.2 Adaptation Actions

More in depth, once the policy has been identified by the Resource Manager, the E2C orchestrator processes it through:

```
rm.perform_action(namespace, prediction, pod_name)
```

This method parses the policy and dispatches one of the following adaptations:

- **Scaling Resources:** For CPU- or memory-scale policies, the orchestrator reads the current *spec.containers[*].resources* requests and limits, computes new values via its scaling functions, and issues a Kubernetes API patch. Because resource limits and requests live in a Pod's spec, and cannot be mutated in place, any change triggers a Pod recreation.
- **Offloading Workloads**: For "task_offloading" or "data_offloading" policies, the task is moved to a node that has been evaluated as being better than the one on which the task is currently running.

Figure 11 illustrates the Task Offloading workflow in action. First, the orchestrator queries real-time load metrics from every node in the cluster and sorts them from highest to lowest utilisation. Then, when a process on an overloaded node exceeds its threshold, the orchestrator selects the least-loaded node from this ordered list and transparently migrates the process there. By redistributing

work in this way, the system relieves hotspots, balances resource usage across the cluster, and maintains optimal performance without manual intervention.

```
INFO:root:*AlertManager rule fired: Pod yolo-performance-dx55w is overloading the node
INFO:root:Fetching metrics for pod yolo-performance-dx55w in ns talon from 1 hours
INFO:root:Request successful
INFO:root:Resource manager predictions: TASK OFFLOADING
INFO:root:Offloading task for pod yolo-performance-dx55w with factor 1.5
INFO:root:Owner detected: job 'yolo-performance'
INFO:root:Best node is {'metric': {'instance': '10.0.0.88:9100', 'internal_ip': '10.0.0.88', 'node': 'k8s-node-1'}, 'value': [1749483290.079
, '0.05647354497351784']}
INFO:root:Target node: k8s-node-1
INFO:root:Re-created Job 'yolo-performance' with nodeAffinity on 'k8s-node-1'
```

Figure 11. Service Orchestrator logs

3.1.3 *Zero-Touch Restart and Rolling Updates*

Regardless of the chosen adaptation, the E2C Orchestrator must restart or delete the affected container to apply the new settings. More in details:

- **Standalone Pods**: The orchestrator deletes and recreates the Pod (e.g. via *kubectl delete pod …* followed by *kubectl apply -f …*).
- **Controller-Managed Workloads:** When the workload is managed by a Deployment, *StatefulSet* or *DaemonSet*, the orchestrator updates the controller's Pod template in its spec. Kubernetes then performs a rolling update: new Pods are spawned with the updated requests/limits, and old Pods are terminated automatically, ensuring zero-downtime.

Should the same alert recur within a brief cool-down interval, the orchestrator transparently escalates to the next-best policy, closing a continuous, self-healing loop that dynamically adapts to evolving cluster conditions.

3.2 Real-time Monitoring and Self-healing

The real-time monitoring and self-healing architecture developed by the TALON project enables continuous supervision of both infrastructure and application behaviour. It detects anomalies when predefined thresholds are exceeded and autonomously executes corrective actions without requiring human intervention. This closed-loop system has been fully deployed and validated in industrial conditions during Pilot 2, relying on a modular stack comprising Prometheus, Grafana, and Node-RED, all integrated within a Kubernetes cluster managed via Rancher. Figure 12 shows an overview of the monitoring-control loop, including the main components involved in each step, from metric collection to automated deployment correction. This diagram provides a high-level view of how **Prometheus, Grafana, Node-RED** and **Kubernetes** interact to ensure real-time self-healing execution.

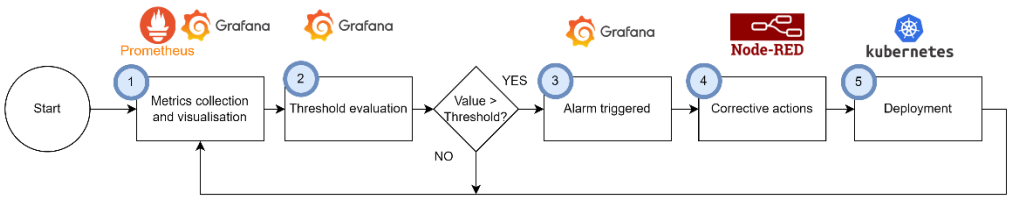

Figure 12. Workflow of the real-time monitoring and self-healing process.

Prometheus is responsible for collecting time-series metrics across the system. It continuously scrapes data related to resource utilisation (CPU, memory, disk, etc.) from monitored endpoints and triggers alerts when specified conditions occur and persist for longer than a defined time window (typically one minute), thereby avoiding responses to transient spikes.

These alerts are visualised in **Grafana**, which offers interactive dashboards to track key metrics in real time, detect trends, and configure alerting logic through PromQL queries. Figure 13 shows the overview of available Grafana dashboards and how they are organised by functional area and resource type.

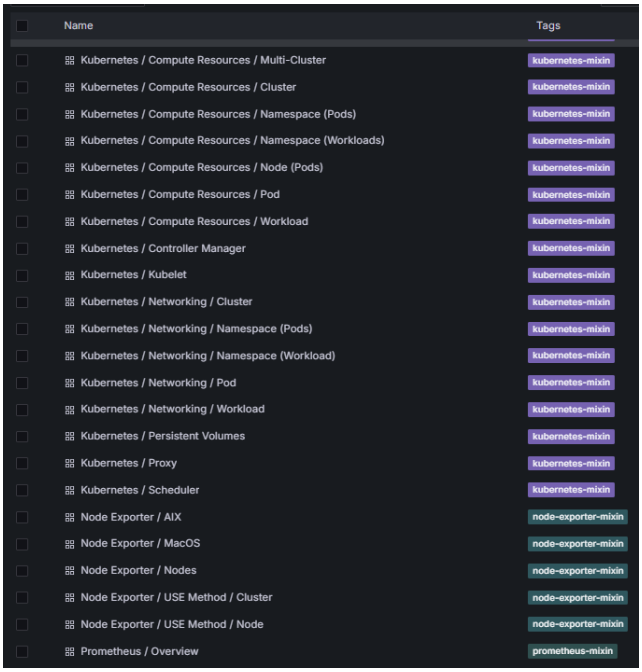

Figure 13. Overview of available Grafana dashboards

To enable automated corrective action, specific alert rules have been configured in Grafana to monitor CPU usage on both control plane and worker nodes. These rules activate when usage exceeds a 70% threshold and trigger a webhook to Node-RED, which initiates a corresponding self-healing flow. The alert rule configuration can be seen in Figure 14, showing the conditions set for triggering automated corrective actions.

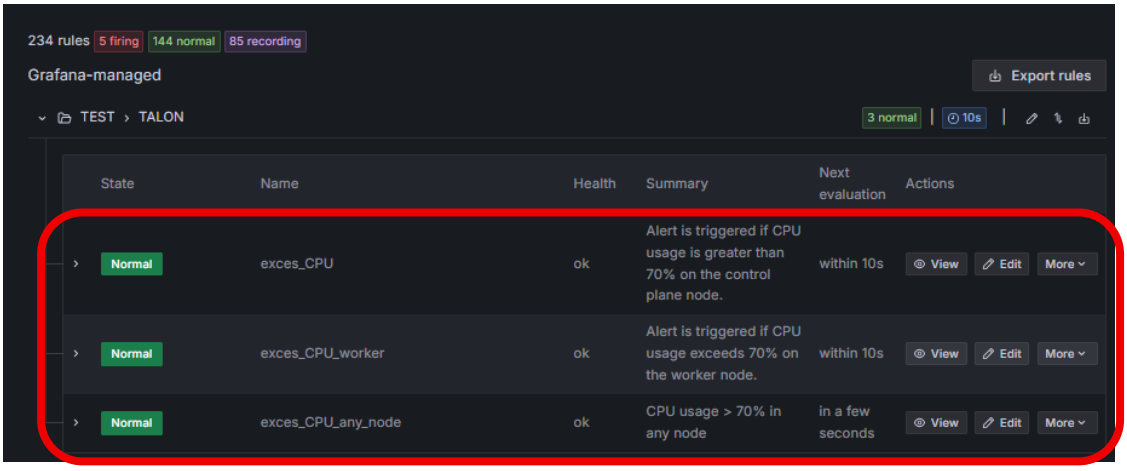

Figure 14. Grafana interface displaying active alert rules for CPU usage thresholds.

Additionally, Figure 14 shows the detailed configuration of one of the system's key alerts, named *excess_CPU_any_node*, which monitors whether any cluster node exceeds 70% CPU usage. This alert is one of several possible conditions that can trigger the automatic execution of corrective flows defined in Node-RED.

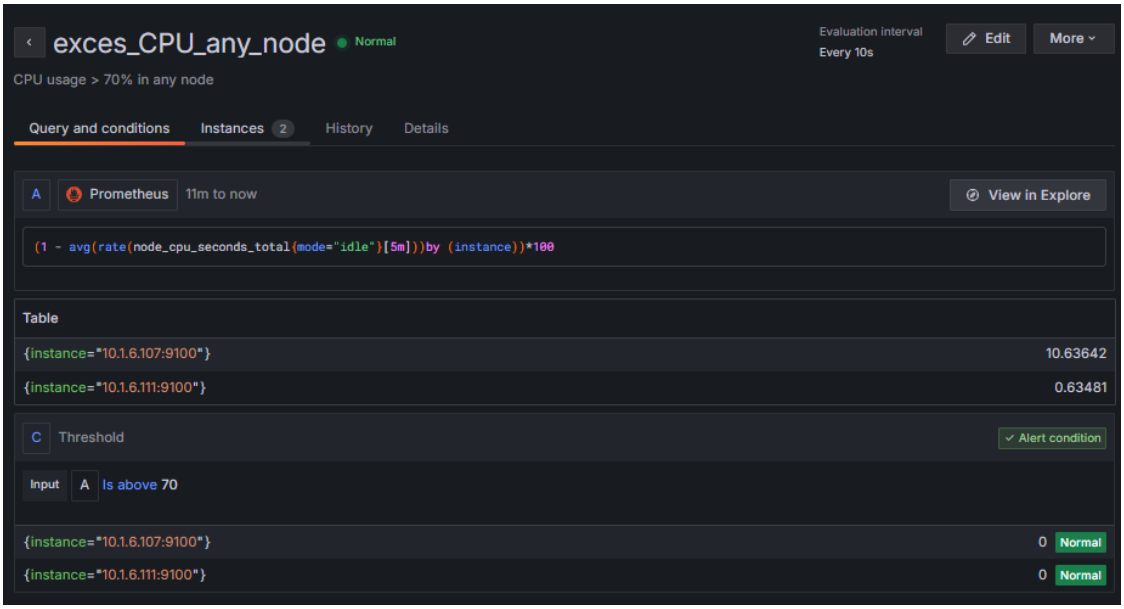

Figure 15. Detailed configuration of the excess_CPU_any_node alert in Grafana

Upon receiving an alert, **Node-RED** processes the message to extract essential data such as the alert type, node affected, severity level, and resource usage. A rule-based flow evaluates whether a remediation action is required and, if so, builds a REST API request to apply changes in the Kubernetes environment. The flows are traceable and include debug nodes for visibility at each step.

Two self-healing rules have been developed and tested:

- **Rule 1**: If high CPU usage is detected in a redundant deployment, Node-RED reduces the number of replicas to alleviate load and stabilise the system. The complete flow is shown in Figure 15, which outlines the execution logic in six stages: receiving the alert, parsing and evaluating the payload, preparing the scaling action, and sending the request to the Kubernetes API via Rancher.
    1. Receive Grafana alerts: An HTTP endpoint (*/worker*) receives a *POST* request containing the alert in JSON format, triggered by a predefined Grafana rule.
    2. Process alerts and extract information: The alert is parsed to extract relevant fields such as the state, alert name, summary, and current CPU value. This data is restructured into a standardised payload.
    3. Decide if action is needed: The logic checks if the CPU usage exceeds the defined threshold. If so, a *scale_down* action is added to the list of operations to be performed.
    4. Prepare action to be sent: A *PUT* request is constructed to interact with the Kubernetes API, including a payload that specifies reducing the number of replicas in the deployment.
    5. Send action to Rancher: The request is authenticated and sent via Rancher, which forwards it to the Kubernetes API to apply the scaling change.
    6. Debug and monitoring outputs: Additional debug nodes are included to display the full alert content, the structured action request, and the API response, ensuring full traceability of the flow.

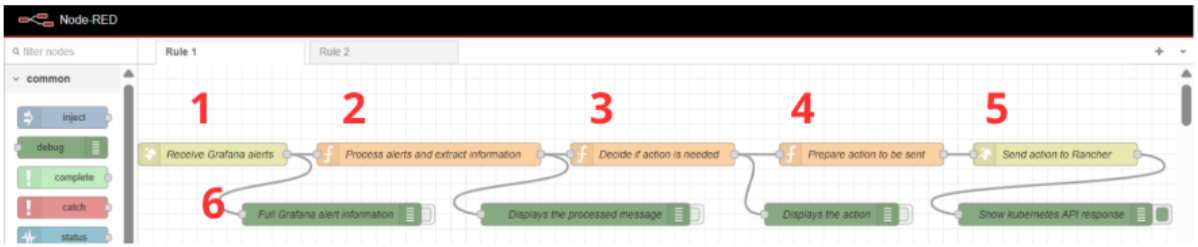

Figure 16. Node-RED flow for Rule 1.

- **Rule 2**: When a specific node becomes overloaded, Node-RED applies a taint to the node to prevent new workloads, deletes the problematic pod, and allows Kubernetes to reschedule it elsewhere. After a 60-second delay, the taint is removed to reintegrate the node. The detailed Node-RED flow for this rule is shown in Figure 16:

    1. Receive Grafana alerts: Alerts are received at the same */worker* HTTP endpoint used in Rule 1, via a *POST* request containing the alert in JSON format.
    2. Extract IP and prepare taint: The flow extracts the instance IP from the alert and maps it to the corresponding node name. It then prepares a PATCH request to apply a *NoSchedule* taint to the node, marking it as unavailable for new Pods.
    3. Apply taint: The request is sent to the Kubernetes API through Rancher, isolating the node from further workload assignments.
    4. Summary information: A summary of the node's metadata (name, taints, labels) is extracted and logged for visibility and traceability.
    5. Delete pod with label: A *DELETE* request is issued to remove the CPU-intensive pod identified by the label app=test2, eliminating the offending workload from the node.
    6. Wait before untaint: A 60-second delay is introduced to allow the orchestrator time to reschedule the pod on a healthier node and stabilise the cluster.
    7. Remove taint: After the delay, a new *PATCH* request is prepared to remove the taint from the node.
    8. Send untaint to Rancher: The untainting request is sent via Rancher to the Kubernetes API, restoring the node's scheduling availability.

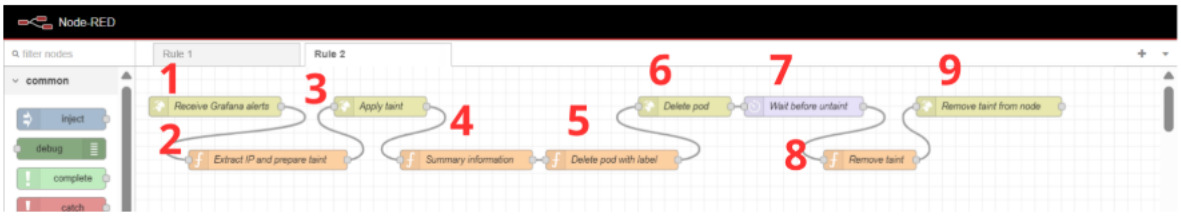

Figure 17. Node-RED flow for Rule 2.

Each rule follows a structured execution sequence: alert reception, data extraction, decision-making, action preparation, API call execution, and debugging. Once deployed, these flows operate autonomously and demonstrate the architecture's capacity for real-time, rule-based corrective response with complete traceability.

In addition to automated self-healing logic, Grafana dashboards offer operators a comprehensive visual summary of system health. CPU, memory, and disk usage across the cluster are aggregated and displayed in intuitive panels, facilitating quick inspection and trend analysis.

Figure 18 presents a consolidated Grafana dashboard showing the cluster-wide status of CPU utilisation, memory usage, and disk consumption, including breakdowns by node. These panels are particularly useful for detecting gradual resource degradation or for conducting root-cause analysis when unexpected behaviour arises.

Figure 18. Aggregated Grafana dashboard for cluster resource utilisation (CPU, memory, disk).

Demonstrations of these self-healing mechanisms, including the real-time triggering and execution of the alert-driven flows, can be viewed in the following video a Canva Presentation.[1]

[1] Demo-WP3-Task 3.5

Conclusion and Future Outlook

Deliverable D3.6 is a pivotal milestone in the TALON project, providing a comprehensive and operational architecture for intelligent, zero-touch orchestration across edge-to-cloud environments. Based on previous technical work, it brings together all the main system components to create a unified platform that can make autonomous decisions, adapt to changing conditions and manage AI workflows reliably and transparently.

It combines cloud-native technologies such as Kubernetes with advanced AI methods like federated learning, LSTM-based decision engines and explainable AI checkpoints. It features a closed feedback loop in which system metrics, service-level objectives (SLOs) and operational context guide automated scaling, offloading and recovery actions. Blockchain support adds transparency to decision-making processes, and energy-efficient AI practices are embedded throughout the system.

TALON has been tested in real-world pilot settings, demonstrating its ability to autonomously manage resources, detect issues and maintain performance within user-defined limits. This confirms the platform's technical readiness and maturity. Nonetheless, some limitations remain. The current system uses fixed, rule-based responses and static AI models that lack continuous learning capabilities. While these are effective in controlled conditions, they struggle to adapt to new or changing environments. Future work will therefore focus on enhancing TALON's ability to predict, learn from, and respond proactively to evolving scenarios.

Key directions for future work include:

- **Predictive and Preemptive Adaptation**: Incorporating machine learning models capable of forecasting failures or overloads, enabling the system to act before disruptions occur.
- **Continuous and Online Learning**: Enhancing the orchestration engine with the ability to retrain or refine its decision models in real time, preventing degradation due to changing workloads.
- **Event-Driven Architecture**: Integrating message-queuing systems (e.g., Kafka or RabbitMQ) to absorb alert surges and decouple orchestration services for improved scalability and fault tolerance.
- **User-Centric Enhancements**: Expanding the TALON Dashboard to offer intelligent suggestions for SLO definitions and to help users interpret orchestration decisions through visual, explainable insights.
- **Expanded Observability and Metrics**: Broadening the scope of telemetry to include user experience indicators, security events, and granular network performance data for better decision-making and resilience.

With this robust architectural foundation, TALON is well-positioned to move into large-scale deployment and evaluation phases. The specifications defined in D3.6 will serve as the backbone for pilot demonstrations, KPI-driven assessments, and the integration of security and business models in subsequent work packages.

More broadly, TALON provides a reference architecture for the next generation of intelligent, self-managing, explainable and energy-aware industrial AI orchestration systems. As Industry 4.0 continues to evolve, TALON's architecture provides a scalable, sustainable approach to managing complex, distributed AI workloads with minimal manual oversight.

References

[1] Sun, T., Li, D., & Wang, B. (2022). Decentralized federated averaging. IEEE Transactions on Pattern Analysis and Machine Intelligence, 45(4), 4289-4301.

[2] Lee, S. H., Sharma, S., Zaheer, M., & Li, T. (2024). Efficient Adaptive Federated Optimization. arXiv preprint arXiv:2410.18117.

[3] Theodorou, G., Karagiorgou, S., & Kotronis, C. (2024, December). On Energy-aware and Verifiable Benchmarking of Big Data Processing targeting AI Pipelines. In 2024 IEEE International Conference on Big Data (BigData) (pp. 3788-3798). IEEE.

[4] Burns, B., Grant, B., Oppenheimer, D., Brewer, E., & Wilkes, J. (2016). Borg, Omega, and Kubernetes. Communications of the ACM, 59(5), 50–57

[5] Harlap, A., Narayanan, D., Phanishayee, A., Seshadri, V., & Ganger, G. R. (2016). PipeDream: Fast and efficient pipeline parallel DNN training. In 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI '16) (pp. 1–15).

[6] Red Hat, Inc. (2020). Kubernetes Patterns: Reusable Elements for Designing Cloud-Native Applications. O'Reilly Media.

[7] Di Francesco, P., Malavolta, I., & Lago, P. (2017). Research on architecting microservices: Trends, focus, and potential for industrial adoption. Proceedings of the IEEE, 105(10), 1909–1930.

Funded by
the European Union

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101070181